Doctoral Dissertation **Academic Year 2012**

Robust High-Performance Real-Time Streaming Control

Kazuhisa Matsuzono

Graduate School of Media and Governance
Keio University

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

High-quality and high-performance real-time video streaming over internet protocol (IP) networks requires 1) maintaining the highest data transmission rate while effectively utilizing network resources and 2) minimizing data packet loss and transmission delay to achieve the best possible streaming quality. However, because current typical congestion controllers based on TCP-friendly rate control (TFRC) prevent occurrences of network congestion reacting susceptibly to packet loss, they cause a significant degradation of streaming quality due to low-achieving throughput and data packet losses.

In this dissertation, motivated by the deployment of wide-area high-speed networks, we propose the two streaming control mechanisms: 1) the dynamic probing forward error correction (DP-FEC) and 2) GENEVA, the streaming control algorithm using generalized multiplicative-increase/additive-decrease (GMIAD). DP-FEC can be utilized for the purposes of IPTV, e-learning and international symposiums that strongly require maintaining the best possible streaming quality. DP-FEC estimates network conditions by successively observing variation in the intervals between packet loss events, and adjusts the degree of FEC redundancy to make a packet loss tolerance as high as possible while minimizing the performance impact of competing TCP flows. GENEVA assumes a usage environment where real-time streaming flows with various data transmission rates and round trip times (RTT) compete (e.g., in the Internet). Using GMIAD mechanism for the adjustment of the degree of FEC redundancy, GENEVA utilizes expected network resources that competing TCP flows fail to copy, and combats bursty packet losses to improve FEC recovery capabilities while cooperating with other competing flows. The evaluation results show that DP-FEC and GENEVA enable high-performance streaming flows to retain higher streaming quality while minimizing an adverse impact on competing TCP performance. The proposed mechanisms can

contribute largely to further growth and promotion of high-performance real-time streaming which is subject to an increasing demand for higher streaming quality.

Keywords :

1. Real-time Streaming, 2. Forward Error Correction, 3. Congestion Control, 4. Internet

Graduate School of Media and Governance, Keio University

Kazuhisa Matsuzono

Thesis Committee:

Supervisor:

Prof. Jun Murai, Keio University

Co-Adviser:

Prof. Hideyuki Tokuda, Keio University
Prof. Osamu Nakamura, Keio University
Dr. Hitoshi Asaeda, National Institute of Information and Communications Technology (NICT)

要旨

Internet Protocol (IP) ネットワーク上での高品位リアルタイム映像配信時には, 1) ネットワーク資源を有効活用し, 2) 輻輳時に発生するデータロス及び遅延に対応することにより達成可能な再生・映像品質維持が強く求められる. しかし, 既存の TCP-Friendly Rate Control を基調とした輻輳制御技術では, ネットワーク資源を有効活用出来ず, 再生・映像品質維持に対する十分な信頼性をエンドユーザに対して提供出来ない.

本研究は, 広帯域ネットワークの普及に伴い, 前方誤り訂正技術 (FEC) を基調とする二つの映像配信制御機構を提案する. Dynamic Probing FEC (DP-FEC) は, 再生・映像品質維持が重視される IPTV や遠隔授業等の環境を想定している. ネットワーク資源を有効活用し, 可能な限り FEC 冗長度を増加させながらパケットロスイベントの発生間隔の変化を観測することにより, 競合している Transmission Control Protocol (TCP) フローに対する FEC の影響度を推測する. FEC によって可能な限りデータロス耐性を向上させ, 競合している TCP フローのパフォーマンス低下を抑えながら再生・映像品質維持を行う. Streaming Control Algorithm using Generalized Multiplicative-increase/additive-decrease (GENEVA) は, 様々なストリーミングフロー (i.e., データ転送レート, ラウンドトリップタイム) が競合するインターネット等の環境を想定している. Generalized multiplicative-increase/additive-decrease (GMIAD) 方式を用いて FEC 冗長度を変更することで, 予測された TCP が利用出来ないネットワーク資源を有効活用し, 他の競合フローと協調しながらバーストパケットロスを抑制することにより, FEC のリカバリ能力を向上させる. NS-2 シミュレータを用いた評価では, 提案した二つの制御機構が TCP フローのパフォーマンス低下を抑え, 高品位リアルタイムストリーミングの品質向上が可能であることを確認した. 本機構の実現により, 今後更なる活性化が見込まれる高品位リアルタイム映像配信に対する信頼性向上が可能となる.

キーワード :

1．リアルタイムストリーミング, 2．前方誤り訂正技術, 3．輻輳制御,
4．インターネット

慶應義塾大学　政策・メディア 研究科

松園　和久

論文審査委員会:

主査:

村井 純,　慶應義塾大学

副査:

徳田 英幸, 慶應義塾大学
中村 修, 慶應義塾大学
朝枝 仁, 独立行政法人 情報通信研究機構 (NICT)

Acknowledgements

I would first like to deeply thank my thesis supervisor, Prof. Jun Murai to give me many useful comments and unique words. Without his recommendation for me to go on to doctoral course, I would not have gained lots of valuable experience in research work. I also had a great and precious time with him in a party of *Murai Adviser Group* which consists of unique and cheerful people like Jun Murai. The time in the group has always encouraged me. I would like to express my gratitude to Prof. Hideyuki Tokuda for his constructive suggestions and supports. His valuable comments with huge background made this dissertation better as well as my master thesis. I am extremely grateful to Prof. Osamu Nakamura, who gave me many insightful comments with bold leadership. Thanks to his direct and straightforward advice, I accomplished to make this dissertation. I am very grateful to Dr. Hitoshi Asaeda, who always provided consultation for me to progress in my paper works as well as my private life. I believe that his dedicated training made my ability improve very much. I am grateful to Prof. Klnam Chon, Keiji Takeda, Assoc. Prof. Hiroyuki Kusumoto, Rodney Van Meter, Keisuke Uehara, Jin Mitugi, and Kazunori Sugiura for thier useful comments about my research work. I also thank to Dr. Achmad Husuni, Kenji Saito, Shigeya Suzuki, Masaaki Sato, Kotaro Kataoka and Ryuji Wakikawa for helping me in advancing my research. Thanks to all the current and past members of Jun Murai laboratory, especially for Tsuyoshi Hisamatsu, Kazuhiro Mishima, Achmad Basuki, Dikshie Fauzie, Hajime Tazaki, Takeshi Matsuya, Yohei Kuga, Kouji Okada and Katsuhiro Horiba. In the end, I would like to thank my father, Hiroaki Matsuzono, and my mother, Masumi Matsuzono for supporting my long student life. They have always encouraged me and given me their devotion. And also, I would like to thank my sister, Eriko Nakamura who have always given me her devotion.

Contents

Abstract ii

Acknowledgement vi

Contents vii

List of Figures ix

List of Tables xii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Research Purpose . . . 3
- 1.3 Outline of Dissertation . . . 4

2 High-Quality Real-Time Video Streaming over IP 5
- 2.1 Background and Problem Statement . . . 5
- 2.2 Requirement . . . 8
- 2.3 Communication Model . . . 9
- 2.4 Related Work . . . 10
  - 2.4.1 Adaptive Packet Scheduling . . . 10
  - 2.4.2 Adaptive Error Control . . . 11
- 2.5 Summary . . . 13

3 Performance Analysis of FEC Schemes 14
- 3.1 Issues about How to Use FEC . . . 15
- 3.2 AL-FEC Codes . . . 16
  - 3.2.1 2D Parity Check Codes . . . 16
  - 3.2.2 RSE Codes . . . 17
  - 3.2.3 LDPC-staircase Codes . . . 17
- 3.3 Implementation of FECFRAME Framework . . . 18
  - 3.3.1 DVTS . . . 18
  - 3.3.2 FECFRAME Framework and FEC Schemes . . . 19
- 3.4 Performance Evaluation . . . 22
  - 3.4.1 Experimental Setup and Performance Metrics . . . 22
  - 3.4.2 Recovery Capabilities . . . 23
  - 3.4.3 Frame Delay . . . 24

3.4.4 Processing Load . . . 26
3.4.5 Discussions . . . 27
3.5 Summary . . . 30

4 DP-FEC: Dynamic Probing FEC 31
4.1 Design Overview . . . 31
4.2 Algorithm . . . 33
4.2.1 Congestion Estimation Function . . . 33
4.2.2 Repeating Congestion Estimation . . . 36
4.2.3 FEC Window Size Adjustment Function . . . 37
4.3 Evaluation . . . 38
4.3.1 Simulation Setup and Performance Metrics . . . 38
4.3.2 Competition with Only Short-Lived TCP Flows . . . 39
4.3.3 Competition with Both Short-Lived TCP Flows and Long-Lived TCP Flows . . . 43
4.3.4 Comparison with Static-FEC Flow . . . 45
4.3.5 Effect of the Value of Threshold FEC Impact . . . 47
4.4 Summary . . . 49

5 GENEVA: Real-time Streaming Control Algorithm using GMIAD 50
5.1 Design Overview . . . 51
5.2 Algorithm and Analysis . . . 53
5.2.1 Acknowledging . . . 53
5.2.2 FEC Source and Repair Transmission . . . 54
5.2.3 FEC Window Size Adjustment . . . 55
5.2.4 Parameter Settings . . . 56
5.3 Evaluation . . . 60
5.3.1 Simulation Setup and Performance Metrics . . . 60
5.3.2 Homogeneous GENEVA Flows VS. TCP Flows in Single Bottleneck Link . . . 62
5.3.3 Heterogeneous GENEVA Flows VS. TCP Flows in Single Bottleneck Link . . . 70
5.3.4 GENEVA Flows VS. TCP Flows in Multiple Bottleneck Links 72
5.4 Summary . . . 74

6 Conclusion 76
6.1 Summary . . . 76
6.2 Future Directions . . . 79

Bibliography 81

Appendix A 90

List of Figures

2.1 Link utilization of bottleneck link capacity when 10 TFRC flows compete with TCP flows. The load of TCP flows was set to 50% of the bottleneck link capacity. The total link utilization keeps about 68% (i.e., about 32% link capacity is underutilized). Because of packet losses caused by TCP flows, each of the TFRC flows cannot maintain the maximum data transmission rate without effectively utilizing network resources. 7
2.2 Communication model. 10

3.1 The DVTS/FECFRAME architecture. 19
3.2 Average data loss rate with 2D, RSE, and LDPC ($k = 170$) codes. . 23
3.3 Average data loss rate with LDPC codes ($k = 500, 1000$). 24
3.4 Frame delay with 2D, RSE, and LDPC ($k = 170$) codes. 25
3.5 Frame delay with LDPC codes ($k = 500, 1000$). 25
3.6 CPU load at the receiver side, with 2D, RSE, and LDPC ($k = 170$) codes. 26
3.7 CPU load at the receiver side, with LDPC codes ($k = 500, 1000$). . 27
3.8 The number of discarded frames when using the equivalent buffer of 100 ms, in the presence of the packet loss probability from 0% to 10%. 28
3.9 The number of discarded frames when using the equivalent buffer of 100 ms, in the presence of the packet loss probability from 10% to 40%. 29
3.10 The number of discarded frames when using the equivalent buffer of 200 ms, in the presence of the packet loss probability from 10% to 40%. 29

4.1 The DP-FEC overview. 32
4.2 Relation between the current calculated loss interval (LI) and maximum acceptable variance of loss interval (ΔLI). 37
4.3 Simulation network model. 38
4.4 DP-FEC performances and average FEC window size under 25% load of TCP flows. 40
4.5 DP-FEC performances and average FEC window size under 50% load of TCP flows. 41
4.6 DP-FEC performances and average FEC window size under 75% load of TCP flows. 42

4.7 DP-FEC performances and average FEC window size in competition with long-lived TCP flows, under 25% load of TCP flows. 44
4.8 DP-FEC performances when the total load of UDP flows (10 streaming flows without FEC) and TCP flows is between 40% and 80%. . 46
4.9 Average loss recovery rate of DP-FEC flows (the cases of *Threshold FEC Impact* of 0.1/0.3 under TCP load of 25%/50%/75%). 47
4.10 TCP-friendliness indexes of DP-FEC flows (the cases of *Threshold FEC Impact* of 0.1/0.3 under TCP load of 25%/50%/75%). 48

5.1 GENEVA overview. 52
5.2 FEC source and repair transmission. 54
5.3 GENEVA scaling properties. 58
5.4 The increment of W_{tot} ($i(W_{tot})$) and decrement of TCP window size (W_{tcp}) as a function of W_{tot}. From the aspect of design that $i(W_{tot})$ should not exceeds the corresponding decrement of $Wtcp$, the constant b (decrement function in GMIAD algorithm) is set to 0.04. . . 59
5.5 The simulation topology (single bottleneck link). 60
5.6 The results of the performance of 10 GENEVA flows under RTTmin 10 ms in competition with long-lived TCP flows. Three metrics (residual data loss rate, the number of bursty packet loss events and TCP performance index) were compared to those of 10 small/large Static-FEC flows and 10 DP-FEC flows under the same network condition. 63
5.7 The results of the performance of 10 GENEVA flows under RTTmin 100 ms in competition with long-lived TCP flows. Three metrics (residual data loss rate, the number of bursty packet loss events and TCP performance index) were compared to those of 10 small/large Static-FEC flows and 10 DP-FEC flows under the same network condition. 64
5.8 Trace of FEC window size of three selected GENEVA flows and packet loss rate on the bottleneck link. 10 GENEVA flows under RTTmin 10 and 100 ms compete with 200 and 700 long-lived TCP flows, respectively. 66
5.9 Trace of FEC window size of three selected GENEVA flows competing with short-lived TCP flows under RTTmin 10 ms, and packet loss rate on the bottleneck link. In this experiment, the load of TCP flows (ρ_tcp) was set to 50% of the bottleneck link capacity, and the rate of an occurrence of TCP flow per second (r_tcp) was set to 25. 68
5.10 Comparison with DP-FEC and Static-FEC performances when the total load of UDP flows (10 streaming flows without FEC) and TCP flows is between 40% and 70%. 69
5.11 Trace of FEC window size of three selected GENEVA flows with different data transmission rates under RTT 10 ms, and packet loss rate on the bottleneck link. 71
5.12 The simulation topology (multiple bottleneck links). 72
5.13 GENEVA performances under the total TCP from 30% to 120%. . . 73

A.1 The number of consecutively lost packet and FEC recovery rate. . . 96

A.2 Packet loss and non-recovery rate in full rate DVTS. 98
A.3 Packet loss and non-recovery rate in half rate DVTS. 99
A.4 Testbed network for evaluating the implementation. 101
A.5 Competition with normal DVTS flow over 100 Mbps link. 102
A.6 Competition with normal DVTS flow over 95 Mbps link. 102
A.7 Competition with normal DVTS flow over 90 Mbps link. 103
A.8 Adaptive DVTS in the recovery of the network congestion. 104

List of Tables

3.1 FEC parameters. 23
3.2 CPU load at the sender side. 27

5.1 The results of 10 GENEVA flows, 10 small/large Static-FEC flows and 10 DP-FEC flows under RTTmin 10 and 100 ms in competition with short-lived TCP flows. 67
5.2 The results of 7 GENEVA flows, 7 small/large Static-FEC flows and 7 DP-FEC flows with different data transmission rates under RTTmin 10 ms, in competition with 200 long-lived TCP flows. . . . 70

A.1 Hardware in our experiment. 97
A.2 Network states and flow types in full rate transmission (30 Mbps). . 100
A.3 Network states and flow types in half rate transmission (15 Mbps). 101

Chapter 1

Introduction

1.1 Motivation

The Internet has been growing by increasing in both access links and backbone networks. According to the State of the Internet published by Akamai Technologies [1], 1) the global average peak connection speed, which implies Internet connection capacity, has been growing, and 2) the speeds in the top 5 countries exceed 30 Mbps. In fact, 1 Gbps broadband access (FTTH) services have been provided with reasonable price in Japan, and 10 Gbps services will be in widespread use in the near future. Thanks to the dissemination of wide-area high-speed networks [52, 53], global IP traffic has increased eightfold over the past 5 years, and it is expected that global IP traffic will reach zettabytes by the end of 2015 [2]. This condition encourages people to take advantage of high-quality and high-performance real-time streaming applications [9, 48] such as interactive e-learning, international symposiums, and telemedicine [75, 5].

On the other hand, although each packet loss ratio measured for various regions has decreased due to the capacity of broadband connections, users still observe around 1% packet loss continuously [3]. Such a small amount of packet loss seriously degrades video quality. For example, as little as 3% MPEG packet loss can cause 30% of the frames to be undecodable [47]. The exponential increase in global IP traffic aggravates the adverse condition even in high-speed networks, because transmission control protocol (TCP) [67, 69] flows (which accounts for

more than 90% of the Internet traffic [63]) induce network congestion by examining maximum available bandwidth, and then leads to *bursty* packet loss [62]. Meanwhile, even if network bandwidth will further grow, such streaming quality degradation will not be fundamentally addressed. Rather, it is a formidable challenge to high-performance streaming applications seeking to optimize their quality without affecting competing traffic flows (e.g., a large amount of TCP flows) in an environment where both network connection speed and IP traffic have been growing.

As one of the major approaches for real-time streaming to deal with network congestion, TCP-friendly rate control [31, 33] has been well studied. In accordance with the definition of TCP friendliness, TFRC tries to maintain fairness with competing TCP flows in the same network condition, while providing a mechanism for a smooth data transmission rate [41, 59]. TFRC reduces the data transmission rate to prevent occurrences of network congestion reacting to packet loss, and increases the data transmission rate while probing for available bandwidth in the absence of packet loss. However, it is well known that in networks with high bandwidth-delay products (BDP), TCP is often found to fail to utilize network resources. The possible lack of a buffer on routers and the existence of concurrent bursting flows prevent TCP flow from effectively utilizing network bandwidth [57]. TFRC in such conditions may unnecessarily force a high-performance streaming flow to reduce the data transmission rate at the expense of video quality. In addition, when the feedback delay, (i.e., the round trip time) is large, a timely and correct estimation of network conditions becomes an impossible task. As a result, unacceptable conditions for video streaming (i.e., packet loss or data rate oscillations) continuously occur [49], and they can have a negative impact on the communication quality of other competing flows (e.g., throughput). In this context, existing approaches do not properly control congestion to fulfill the demands of end-users wanting the highest streaming quality. In networks where the physical bandwidth is relatively larger than the total consumption bandwidth of high-performance streaming flows, senders maintaining the highest data transmission rate would not make a severe impact on the congestion [58]. Thus, high-performance streaming flows using TFRC suffer from a significant degradation of streaming quality due to low-achieving throughput and bursty packet loss.

1.2 Research Purpose

In this dissertation, we focus on the scenario of multiple coexisting flows along a high-speed network path, where both high-performance real-time streaming flows and TCP flows are competing. The key assumption in our research is that network congestion as indicated by packet loss occurs mainly due to TCP flow, where the bottleneck link bandwidth is notably larger than the total consumption bandwidth of high-performance streaming flows sending data packets at the maximum rate (i.e., at the maximum audio/video quality). Our research purpose is to provide new effective controllers for high-performance streaming applications to achieve the best possible quality in an environment where both network connection speed and IP traffic have been growing.

To tackle the problem that an existing congestion controller fails to maintain higher streaming quality for high-performance real-time applications in high BDP networks, we propose the two algorithms using Forward Error Correction (FEC), 1) the dynamic probing forward error correction (DP-FEC) and 2) GENEVA, the streaming control algorithm using generalized multiplicative-increase/additive-decrease (GMIAD). DP-FEC provides a promising method for achieving higher streaming quality while seeking the behaviors of competing TCP flows. DP-FEC estimates the network conditions by dynamically adjusting the degree of FEC redundancy while trying to recover lost data packets. By successively observing variation in the intervals between packet loss events, DP-FEC effectively utilizes network resources. It aggressively increases the degree of FEC redundancy to make a packet loss tolerance as high as possible while minimizing the performance impact of competing TCP flows. GENEVA also provides a promising method for achieving higher streaming quality by adding redundant data packets while both effectively utilizing network resources. Assuming that real-time streaming flows with various data transmission rate and round trim times (RTT) compete, GENEVA combats bursty packet losses through adjustment of the transmission rate using the GMIAD mechanism. GENEVA avoids low-achieving throughput by allowing the streaming flows to maintain moderate network congestion, and also considers the effect on the performance of other competing flows. The GMIAD mechanism adjusts the consumption bandwidth by changing the degree of redundant data packets to suppress bursty packet loss, which contributes to reliability

and stability of the streaming playback quality. In addition, it tries to effectively utilize expected available bandwidth that competing TCP flows cannot consume due to reductions in the transmission rate in response to packet loss. Since we apply FEC to the proposed algorithms, we bring out FEC performance by investigating actual three FEC schemes. Then, we evaluated the two algorithms using an NS-2 simulator [78]. The results show that according to user usage environments, DP-FEC and GENEVA enables high-performance streaming flows to retain higher streaming quality while minimizing an adverse impact on competing TCP performance.

1.3 Outline of Dissertation

This dissertation is organized as follows. Chapter 2 details the problems for high-performance real-time streaming and examines the requirements, and describes related works. Chapter 3 makes a performance analysis of FEC schemes using FECFRAME framework [77]. Chapter 4 describes the DP-FEC design and algorithm, and evaluates the effectiveness of DP-FEC. Chapter 5 describes the GENEVA design and algorithm, and evaluates the effectiveness of GENEVA. In Chapter 6, we present our conclusions.

Chapter 2

High-Quality Real-Time Video Streaming over IP

2.1 Background and Problem Statement

Real-time streaming applications commonly rely on the unreliable transport services provided by user datagram protocol (UDP) [68] that is a fast and lightweight protocol. Thus, video transmission encoded for real-time streaming generally needs to keep pace with changes in network conditions. To escape from interruptions and stalling, streaming applications compensate for packet loss by various approaches whose suitability could be dependent on encoding techniques and communication environments. High-quality real-time video streaming has strict requirements on interactivity (i.e., low perceived latency) and packet loss (i.e., high perceived playback quality). To keep both interactivity and video quality as high as possible, high-quality real-time streaming prefers to use low video compression if available network bandwidth is larger than the required consumption bandwidth. This makes it difficult to dynamically reduce the data transmission rate to avoid packet loss by changing the encoding parameters at the sender side according to network conditions, because it may need more time to encode/decode and result in a degradation of playback quality by buffer underrun at the receiver side. Thus, network heterogeneity and congestion make it more difficult for high-performance real-time

video streaming to properly adapt to changes in network conditions for improving the streaming quality.

Most of the proposed congestion control mechanisms for real-time video streaming applications try to achieve TCP-friendliness according to its definition [4, 31, 59, 41, 32]; TFRC thus enables a streaming flow to maintain the desired smoothness of data transmission rate and fairness with coexisting TCP flows. However, because high-performance video streaming requires a large amount of bandwidth that cannot be utilized by TCP, throughput regulation based on TCP-friendliness forces the streaming flow to reduce the data transmission rate at the expense of video quality (i.e., the transmission rate is adjusted by changing the parameters for encoding the video), which causes low network utilization. Video quality is increasingly reduced as bandwidth delay products (BDP) increase, because TCP is inefficient under high BDP environments. A high-performance TCP protocol (such as Scalable TCP [36] and High-speed TCP [37, 38]) could be used in real-time streaming, but it typically results in some quality degradation (such as a longer startup delay and lack of smoothness). As another critical problem, since the window controls that competing TCP flows adopt send data packet in bursts (and may have sent a large number of packets by the time the sender learns of an occurrence of network congestion), streaming flows suffer from bursty data packet losses (i.e., consecutively lost data packets). In addition, since TFRC examines packet loss conditions as an indicator of network congestion, it often delays the action for rate control, which can lead to a situation of continuous data packet loss.

To clarify the specific problem of competition between TFRC streaming flows and TCP flows, we used NS-2 to simulate a single bottleneck with a 1Gbps link capacity. The round-trip propagation delay was set to 10 ms. DropTail queue was used and the queue size was set to a value equivalent to BDP. To create network congestion, we generated short-lived TCP flows (TCP-Sack) using a Poisson process with an average rate of 12.5 flows per second (r_tcp), and the size of a TCP flow (s_tcp) follows a Pareto distribution with an average of 5 MB and a shape parameter of 1.5. We define the load of TCP flows as $\rho\_tcp = r\_tcp \times s\_tcp$. The load of TCP flows thus becomes 50% of the bottleneck link capacity. We used 10 TFRC streaming flows with a maximum data rate of 30 Mbps. Each simulation ran for about 120 sec.

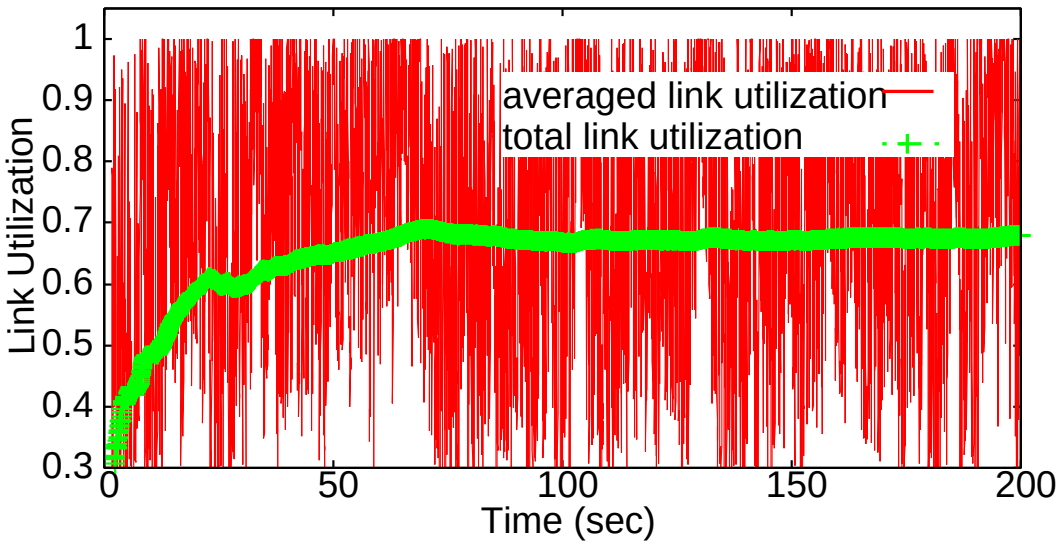


Figure 2.1: **Link utilization of bottleneck link capacity when 10 TFRC flows compete with TCP flows. The load of TCP flows was set to 50% of the bottleneck link capacity. The total link utilization keeps about 68% (i.e., about 32% link capacity is underutilized). Because of packet losses caused by TCP flows, each of the TFRC flows cannot maintain the maximum data transmission rate without effectively utilizing network resources.**

Fig. 2.1 shows the link utilization averaged over Round Trip Time (RTT) interval and also its total link utilization. Since an increasing congestion window size during TCP slow-start or congestion avoidance phase continuously causes packet losses, the competing TFRC flows cannot maintain the maximum data rate, and the overall average throughput of the TFRC flows becomes about 19.1 Mbps. Although the TFRC flows reduce the data transmission rate in response to packet loss, 32% of link capacity is underutilized (the total link utilization keeps about 68% after 65 seconds) and the average data loss rate becomes about 0.5%. Because of packet losses cause by TCP flows, a TFRC flow with relatively high available bandwidth suffers from a quality degradation due to both reductions in the data transmission rate and data packet losses. TFRC thus overemphasizes the maintenance of TCP-friendliness too much, which leads to low network utilization. From this viewpoint, it is important to protect playback quality from packet loss while preserving the highest data transmission rate and executing effective congestion

control to better utilize network resources without interacting badly with existing TCP traffic.

2.2 Requirement

As described in Section 2.1, high-performance real-time streaming using TFRC suffers from bursty packet losses that competing TCP flows induce, and unnecessary reductions in the data transmission rate (i.e., video quality) due to improper congestion control. Existing congestion control protocols thus emphasize the maintenance of TCP-friendliness too much, which leads to low network utilization especially in high BDP links. Instead of using TFRC, an alternative protocol is needed for high-performance real-time streaming to avoid a degradation of playback quality caused by packet losses and to preserve the highest data transmission rate in congested networks, when the physical network bandwidth is much larger than the total consumption bandwidth of high-performance streaming flows. When the physical bandwidth is notably less than the total consumption bandwidth of high-performance streaming flows, data senders should reduce the data transmission rate at the cost of video quality. In this dissertation, we do not assume such a situation. From this viewpoint, it is important to protect playback quality from packet loss while executing effective congestion control to better utilize network resources without interacting badly with existing TCP traffic.

Forward Error Correction (FEC) is the well-known algorithm that has been notably used in streaming applications to improve playback quality in the presence of data packet losses. An application level forward error correction (AL-FEC) approach prevents retransmission delays by preventively adding redundant packets to the streaming data flow. Thanks to the redundancy, a certain number of missing data packets can be recovered. However, ascertaining and controlling optimal FEC redundancy in congested networks is a real challenge, because 1) it is difficult for a sender to determine the packet loss pattern at each moment (as there is a feedback delay) and to predict the future packet loss pattern, and 2) increasing FEC redundancy may disturb both streaming and competing flows when the conditions of competing flows are sensitively oscillated in the network. The following points thus must be considered when enhancing FEC redundancy in congested networks:

- TCP flows tend to transmit their packets in bursts especially in high BDP links, and therefore may cause bursty packet losses [62]. Because this situation often makes it difficult to appropriately adjust FEC redundancy (such that FEC cannot recover lost data packets), an occurrence of bursty packet loss should be suppressed as much as possible to optimize the recovery of lost data packets.

- Controlling FEC redundancy is not inconsequential to the behavior of other flows. It is important, therefore, to consider the impact of FEC on the communication quality of other flows while controlling FEC redundancy. In this case, since the flow controlling FEC redundancy cannot behave similarly to TFRC or current TCP, existing TCP-friendliness cannot be maintained. However, to minimize the adverse effect on TCP performance as much as possible, the degree of FEC redundancy should be adjusted adequately.

2.3 Communication Model

To simplify the requirements for controlling FEC redundancy according to network and user usage environments, we assume that high-performance real-time application transmits data packets in 1) single bottleneck link or 2) multiple bottleneck links as shown in Fig. 2.2.

In single bottleneck link, streaming flows have almost the same propagation delay and share one bottleneck link. That is the case primarily for interactive e-learning or video conference in intercampus networks and IPTV service [50]. In this situation in which streaming and TCP flows compete over relatively simple and managed networks, service providers and users tend to relatively emphasize a focus on maintaining the best possible streaming quality. On the other hand, in multiple bottleneck links, streaming flows have different propagation delays and share some bottleneck links. That is the case primarily for real-time video transmission over the Internet. In this situation, it is important to minimize an adverse effect on TCP performance [29]. According to the difference of the requirements, we propose the two mechanisms, 1) DP-FEC, the dynamic probing forward error correction for single bottleneck link model, and 2) GENEVA, the streaming control algorithm using generalized multiplicative-increase/additive-decrease (GMIAD) for multiple

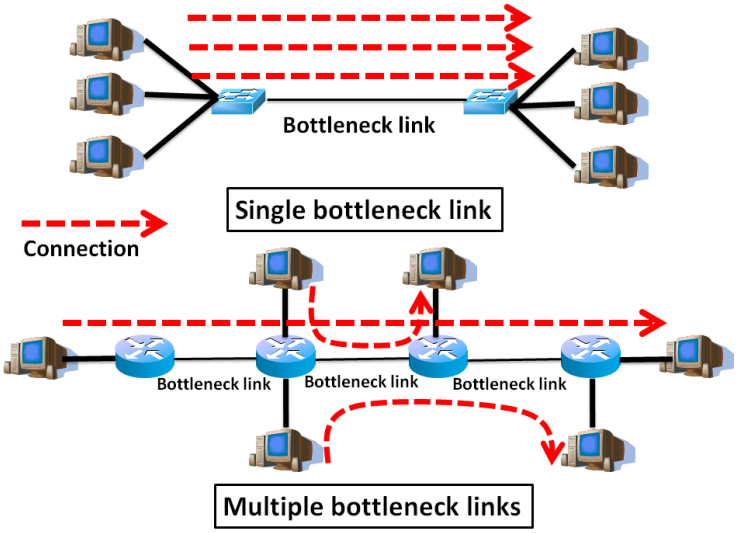


Figure 2.2: **Communication model.**

bottleneck links model.

2.4 Related Work

In this section, we describe related works in terms of adaptive packet scheduling and adaptive error control.

2.4.1 Adaptive Packet Scheduling

Many congestion control mechanisms for real-time streaming have been proposed and analyzed. Rejaie, et al. [4] developed a rate-based congestion control mechanism that employs an additive-increase, multiplicative-decrease (AIMD) algorithm [70] to achieve TCP-like behavior. The datagram congestion control protocol (DCCP) [55, 56] was proposed by Kohler et al. to provide TCP-friendly rate control [31, 33]. The mechanism function is designed to behave fairly with coexisting TCP flows in terms of the consumption bandwidth. Papadimitriou et al. have proposed a rate control mechanism from the perspective of inter-protocol fairness [32]. Using a packet loss as a congestion indicator, this mechanism decreases

the data transmission rate by a previously determined degree. If no congestion is sensed, it periodically increases the data transmission rate in incremental steps. Feng, et al. [59] defined the new TCP friendliness necessary to achieve a desired constant data rate while maintaining fairness with coexisting TCP flows. However, because these mechanisms utilize packet loss as a congestion indicator and severely reduce the data transmission rate in response, they are not suitable for high-quality real-time video streaming that needs to maintain a higher data rate and avoid a degradation of playback quality, as described in Section 2.1.

2.4.2 Adaptive Error Control

Various numerical studies, using actual experimental measurement or analytical models, have evaluated the effectiveness of FEC. Bolot, et al. [46] measured audio packets transmitted via the Internet, and analyzed the packet loss patterns. The results suggested that FEC is effective for low bandwidth streaming, such as an audio application, even when the network conditions are unstable because of network congestion. However, Altman, et al. [51] proposed a detailed queuing analysis based on a ballot theorem, and concluded that FEC caused only a slight improvement in the audio streaming quality under any amount of redundant FEC data. In [10], an analysis of FEC effectiveness in ATM networks indicated that performance gains quickly diminish when all traffic sources employ FEC and the number of sources increases. Shacham, et al. [11] used both analytic and simulation models to evaluate a situation using redundant parity packets, residual packet-loss rates after FEC decoding were reduced by up to three orders of magnitude. However, Xunqi, et al. [12] indicated that the simplified analysis in [11] overestimated the performance of FEC because it assumed that the packet loss process is independent. As an alternative approach, Xunqi, et al. [12] proposed a recursive algorithm based on the algorithm proposed by Cidon, et al. [13] to compute block error density, and analyzed FEC performance in various FEC parameters (i.e. the encoding length, code rate, and interleaving depth). They concluded that an interleaving method increases FEC effectiveness for real-time streaming over congested networks. Although these numerical results vary depending on the loss model that is used or the testbed network, they provide insights into FEC recovery capability that can be applied to audio and video streaming.

A number of mechanisms for adjusting the degree of FEC while cooperating with TFRC have been proposed. To improve the video and playback quality of MPEG video while maintaining fairness with competing TCP flows, Wu, et al. [47] developed an optimization mechanism that adjusts FEC redundancy and the data transmission rate under bandwidth constraints of the TFRC equation. Because the proposed mechanism depends largely on the TFRC throughput equation, it tends to suffer from a degradation of streaming quality without effectively utilizing network resources in higher BDP environments. On the other hand, GENEVA is not subject to TFRC rate to avoid a degradation of video quality while effectively utilizing available network resources by adjusting the FEC window size. Seferoglu, et al. [41, 42] focused on packet losses caused by TCP-induced congestion, and proposed the TFRC that decides the best allocation of the available TFRC rate between source and FEC packets, based on the congestion prediction derived by the correlation between packet losses and the estimated RTT fluctuation. However, such predictions of loss events by using delay information result in a severe quality degradation, especially in high bandwidth paths [39, 40].

On the other hand, using packet loss characteristic modeled with two-state Gilbert model, Bolot, et al. [76] proposed the mechanism that determines the degree of FEC together with TFRC. In the presence of packet losses due to channel errors rather than network congestion, FEC techniques and its effectiveness for end-to-end protocols were examined, and the adaptive FEC schemes were proposed to combat such packet losses [43, 44]. Kondo, et al. [54] proposed FEC method for high-quality video transmission that uses packet interleaving to address bursty packet losses and simply decides the degree of FEC redundancy based on only observed packet loss rates. For video applications using TCP, Tsugawa, et al. [8] applied an adaptive FEC scheme to TCP in order to avoid throughput fluctuations which result in video quality degradation. The scheme utilizes the algorithm proposed in [19] and derives the congestion window size needed for achieving the required data transmission rate. It then determines the appropriate degree of FEC to maintain the required rate according to the packet loss conditions, which achieves higher performance than static FEC approach. The adaptive rate control with Dynamic FEC mechanism was proposed [66]. This mechanism tries to estimate network conditions while recovering lost data packets by fully added FEC redundancy. Since it focuses on the quality improvement for its own flow by ag-

gressively increasing FEC, it holds the potential for adversely increasing traffic congestion and disturbing other communication qualities.

2.5 Summary

In this chapter, we described the basic technology behind high-performance real-time streaming applications, and detailed the problems for them. We indicated that high-performance real-time streaming flows with an existing congestion controller suffer from a quality degradation due to both reductions in the data transmission rate and data packet losses even when relatively high available bandwidth for competing flows exists. Then, we examined the requirements and described related works. In the next chapter, we make a performance analysis of FEC schemes, and describe the results.

Chapter 3

Performance Analysis of FEC Schemes

As described in Section 2.1, real-time streaming applications typically require minimizing packet loss and transmission delay so as to keep the best possible playback quality. From this point of view, IP datagram losses (e.g. caused by a congested router, or caused by a short term fading problem with wireless transmissions) have major negative impacts. Although Application Layer Forward Error Correction (AL-FEC) is a useful technique for protecting against packet loss, the playback quality is largely sensitive to the AL-FEC code/codec features and the way they are used. As we apply AL-FEC to the proposed algorithms, it is of considerable importance to verify the AL-FEC effectiveness when it is applied to a high-performance, high throughput real-time video streaming application.

In this chapter, we consider three FEC schemes for the erasure channel: 2D parity check codes, Reed-Solomon over GF(2^8) codes, and LDPC-Staircase codes, all of them being currently standardized within Internet Engineering Task Force (IETF) [34]. We have integrated these FEC schemes in the FECFRAME framework, a framework that is also being standardized at IETF, and whose goal is to integrate AL-FEC schemes in real-time protocol stacks in a simple and flexible way. We modified the Digital Video Transport System (DVTS) high-performance real-time video streaming application [9] so that it can benefit from FECFRAME in order to recover from transmission impairments. Then, we carried out several

performance evaluations in order to identify, for a given loss rate, the optimal configuration in which DVTS performs the best.

3.1 Issues about How to Use FEC

Packet loss resilience can be achieved in two ways: thanks to retransmissions or thanks to the addition of redundancy. The Automatic Repeat reQuest (ARQ) scheme, although widely-used in many transmission protocols (e.g. TCP), needs at least one Round Trip Time (RTT) to recover from a loss. This is an issue if this RTT exceeds the maximum acceptable delay of a real-time stream. Another problem is the fact that ARQ does not scale well in case of multicast or broadcast diffusion. On the opposite, Application Layer Forward Error Correction (AL-FEC) codes rely on error correcting codes for the packet erasure channel. With this type of channel, packets are either received without any error or totally lost during the transmission. The use of AL-FEC codes prevents retransmission delays by preventively adding redundancy packets to the streaming data flow. Thanks to this redundancy, a certain number of missing source packets can be recovered at the receiver side without the need to ask the sender for lost packets. Therefore AL-FEC is commonly used for real-time streaming applications like a video conference systems.

More precisely, an AL-FEC encoder adds $n - k$ repair packets to each block of k source packets, which allows a receiver to rebuild all of the k source packets if more than k packets are received among the n packets sent. Maximum-Distance Separable (MDS) codes, such as Reed-Solomon codes [15, 16], can recover all missing packets from any set of exactly k packets, while LDPC (Low Density Parity Check) codes [17, 18] need to receive a small number of extra packets in addition to k. The performance of AL-FEC codes and the associated codecs (their software implementation), in terms of recovery capability and encoding/decoding delay largely varies, depending on the code intrinsic properties (e.g. is it an MDS code or not?), on the details of the codec (e.g. which kind of decoding algorithm is used, in particular with LDPC codes), the way transmissions occur (e.g. are packets sent sequentially or in a random order?), and the code parameters (e.g. the encoding block length n and the code rate k/n). Since real-time applications

are sensitive to both delay and packet losses, and since these two aspect are contradictory (the larger the block size, the better the protection, but the higher the delay), it is usually necessary to find a trade-off.

In the past, besides the diversity in the underlying FEC encoding/decoding techniques, different FEC mechanisms are proposed and analyzed to improve the performances. For instance, the impacts of packet scheduling and loss distribution on FEC performances are studied [20]. In [21], four different FEC techniques, that utilize `XOR` (eXclusive-OR) and RSE erasure codes, are adopted according to the network conditions. In [22] the performances of the LDPC-staircase codes are studied when using a hybrid IT/ML decoding scheme, both from the erasure recovery capability and decoding complexity viewpoints. This work is one of the foundations of the current paper, with the difference that in our work we are considering real-time flows, with different goals. Concerning the parameter adjustment techniques (e.g. code rate), many studies have been conducted so far [47, 76, 66]. However, in the context of high-performance real-time applications, practical results and evaluations as a function of the FEC codes have not been fully studied. Many past studies rely on streaming software featuring a low bit-rate (i.e. less than 10 Mbps) and non-real-time flows (i.e. non strictly enforced). In that case phenomena like the impact of the codec in terms of CPU load are not visible.

3.2 AL-FEC Codes

Let us focus now on the three AL-FEC codes being considered in our work:

3.2.1 2D Parity Check Codes

The 2D parity check codes (2D codes) are classified as "simple codes" [23], which suggests that these codes are effective for recovering packet losses under very low loss conditions. For example, a simple way of adding protection against a single packet loss consists in creating a parity (`XOR`) of a certain number of source symbols (where a symbol is a fixed size unit of data from the AL-FEC encoder/decoder point of view). The 2D codes are only slightly more complex than that: the k source symbols are arranged in a $p \times p$ square matrix, where $p = \sqrt{k}$ (we assume

that p is an integral value). Then, for each row (resp. each column) a single repair symbol is created by computing the XOR sum for all source symbols in the row (resp. column). We can rebuild any row (resp. column) of the matrix from any p out of $p+1$ symbols in the row (resp. column). Of course these codes have limits: they are not MDS codes (there are situations where k received symbols do not enable decoding to succeed), and they create constraints on both k ($p = \sqrt{k}$ must be an integral value) and on the number of repair symbols, $2p$ (said differently the code rate is necessarily equal to $\frac{k}{k+2p}$).

3.2.2 RSE Codes

The Reed-Solomon codes for the Erasure channel (RSE) [15, 16] are classified as "small block codes" [23] (i.e. k, n are limited) and they are one of the most popular FEC codes. RSE codes are intrinsically limited by the Galois Field it uses (e.g. with GF(2^8), $n \leq 2^q - 1 = 255$). Since the cost of the finite field operations quickly increases with the finite field size [15], most of the practical applications restrain themselves to GF(2^8), even if it also limits n and k parameters. Yet, RSE codes are MDS codes which is a major asset (lost source packets can be recovered as soon as *exactly* k packets out of n are received for this block). In this dissertation we will focus on RSE codes over GF(2^8) only.

3.2.3 LDPC-staircase Codes

The LDPC-staircase codes [24, 17, 18] are classified as "large block codes" [23] (i.e. k, n can be very large) and they are a particular case of regular Repeat Accumulate codes. Although they are not MDS codes, they have many advantages like the possibility to operate efficiently with large source k and n parameters, with a linear time encoding complexity. The parity check matrix H of LDPC-staircase codes is a $(n-k) \times n$ binary matrix which can be divided into two parts: the left side of the matrix, H_1, is composed of $n-k$ rows for k columns (i.e. the source symbols), while the right side of the matrix, H_2, is composed of $n-k$ rows for $n-k$ columns (i.e. the parity symbols). Here the H_2 matrix has a "staircase" (double diagonal) structure. It means that each parity symbol is the XOR sum of the previous parity symbol plus a very small number of source symbols. The H_1

matrix is filled in a fully regular way as follows:

1. Insert N_1 "1-s" randomly but evenly into each column;
2. Check there are at least two "1-s" per row. If not, add one or two extra "1-s" randomly to these rows.

In [25, 22], a hybrid ITerative/Maximum Likelihood (IT/ML) decoding was proposed to improve the erasure recovery capabilities (now close to that of MDS codes) while keeping a moderate decoding complexity. This algorithm combines the optimal correction capabilities of the ML decoding with the low complexity of the IT [26] decoding. It works as follows: start decoding with the linear complexity IT algorithm. If IT decoding fails and if it is known that no additional symbol will be received, switch to the more complex ML decoding (basically a Gaussian Elimination) and try to finish decoding. It is clear that a receiver can decide if and when ML decoding is used, depending on local criteria (e.g. battery or CPU capabilities), independently from other receivers, which is a great asset. In order to have good erasure recovery performances under IT/ML decoding, even with small n values, we adjust the N_1 parameter as follows [22]: if $n \leq 2^8$, $N_1 = 7$, otherwise $N_1 = 5$.

3.3 Implementation of FECFRAME Framework

We now describe the FECFRAME framework that is integrated into DVTS [74, 9] (Fig. 3.1).

3.3.1 DVTS

The DVTS application can send and receive DV (Digital Video)/HDV (High-Definition Video) data over RTP [27]. The DV camera generates the DV/HDV stream, sends it over the IEEE 1394 interface to the sending host that runs the DVTS sending application. This latter encapsulates the DV/HDV frames into RTP packets and transmits them over the IP network. At the remote host, DVTS application upon receiving a DV/RTP or HDV/RTP packet, attaches an IEEE 1394 header to the received or reconstructed DV/HDV packets and transfers them

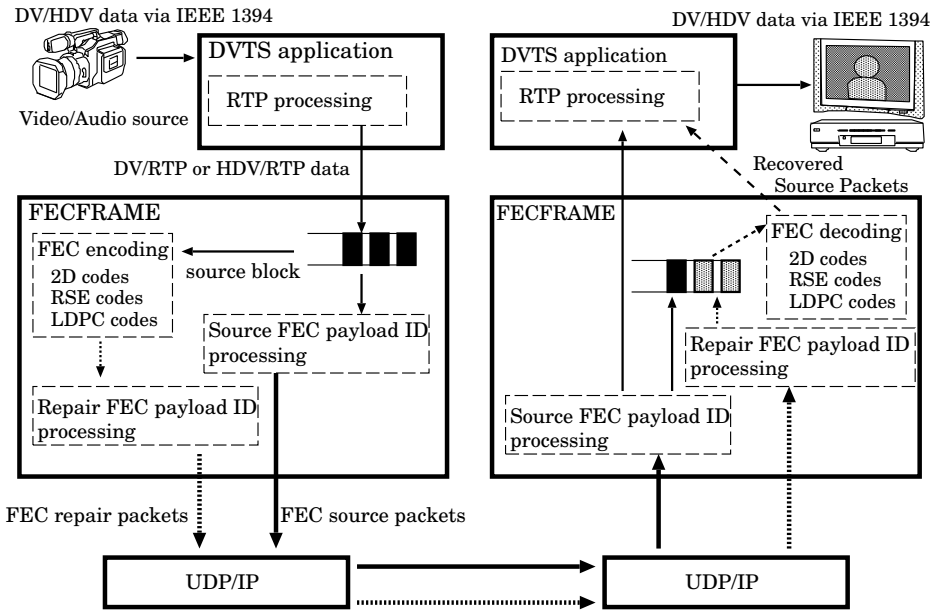


Figure 3.1: **The DVTS/FECFRAME architecture.**

to the DV/HDV recorder deck via the IEEE 1394 interface. A full DV stream consumes over 30 Mbps with standard NTSC quality video (525 lines and 29.97 fps). In case of HDV format (i.e. MPEG-2 TS), the full stream consumes about 25 Mbps.

3.3.2 FECFRAME Framework and FEC Schemes

The FECFRAME framework [77] defines a generic way of plugging FEC schemes (i.e. the codec implementing an AL-FEC code) dynamically into a real-time (or non real-time) protocol stack, for instance between the RTP and UDP layers. Thanks to this framework, a sending streaming application can generate and send FEC repair packets (i.e. redundancy), and a receiving streaming application can recover from lost source packets and pass them to the application as if they had been received normally. Several parameters need to be adjusted and communicated to both the sender and the receiver: 1) the nature of the FEC scheme (we consider 2D codes, Reed-Solomon and LDPC-staircase in this work), 2) FEC scheme specific parameters (e.g. with Reed-Solomon the finite field size, $m = 8$), 3) the symbol

length (E), which is either fixed for the whole duration or adjusted per block, depending on the FEC scheme, 4) the source block length (k), 5) and the code rate k/n, or equivalently the total number of encoding symbols (source plus repair), n (subject to some constraints depending on the FEC scheme).

The encoding procedure, at the sender side, is as follows:

1. **Source block creation:** The FECFRAME instance receives (H)DV/RTP packets (known as Application Data Units, or ADU) from the application and stores them in a source block buffer for future encoding. To each ADU three bytes are prepended to form the source symbol (the source symbol is the unit of data used during the encoding/decoding process): 1 byte for the "flow ID" and 2 bytes for the ADU length. Then, since all source symbols need to be E bytes long, zero padding is added at the end of each ADU when needed. The Flow ID contains an integer that identifies the source flow (or application) to which this ADU belongs. This is necessary when several flows (several applications) are protected together by the same FECFRAME instance. The length contains an integer that indicates the ADU length, in bytes, without the three bytes and padding. This is necessary during decoding to strip padding.

2. **FEC encoding:** When the number of symbols stored in a source block buffer reaches k, or when the real-time constraints prevents the sender from waiting for additional ADUs (it can happen e.g. with variable bit rate flows), FEC encoding takes place. A certain number of repair symbols are created for this block, in accordance with the code rate (equivalently n) limitation.

3. **"Source" and "Repair FEC Payload ID" addition:** Then a "source FEC payload ID" (case of an ADU) is added at the end of the (H)DV/RTP packet and transmission takes place. A mode exists where the (H)DV/RTP packet is sent without any modification (no "source FEC payload ID") for backward compatibility purposes. This mode has limitations and will not be considered. Similarly a "repair FEC payload ID"(case of a repair symbol) is prepended (rather than appended) to each repair symbol, and transmission takes place. The goal of these FEC payload ID are, for a receiver, to identify which block the symbol belongs to as well as the symbol "position" inside

this block. The receiver can therefore gather all the symbols belonging to the same block and launch decoding if needed.

At the receiver side, the FECFRAME instance receives both FEC source and repair packets from the network. The decoding procedure is as follows:

1. **FEC source and repair packet storage:** Upon receiving a FEC source packet, the FECFRAME instance 1) passes a copy of it, without the source FEC payload ID, to the application immediately. It also 2) keeps a copy of it with other packets that belong to this block. Since the RTP data is immediately passed to the application, the application benefits from it without any additional waiting time. Upon receiving a FEC repair packet, the FECFRAME instance stores this packet in the buffer associated to the block it belongs to.

2. **FEC decoding:** If some source packets are missing in a source block and if the number of repair symbols received for that block is sufficient, FEC decoding is launched. Depending on the FEC code being used, it works as follows:

   - *2D codes:* When a packet belonging to the next source block arrives, a decision is taken: if at least k symbols are available for the current block, decoding is launched. If we ignore possible packet reordering, this is the sign that no additional packet will be received for the current block. With possible packet reordering, the receiver should wait some more time.
   - *RSE codes:* FEC decoding is launched as soon as exactly k source and repair symbols are received.
   - *LDPC-staircase codes:* LDPC-staircase codes require slightly more than k symbols for decoding to succeed. Based on experimental results [22], we set the threshold at $k \times (1.05)$, which corresponds to a very high decoding success probability and a low decoding complexity (IT decoding will significantly simplify the linear system to be solve during ML decoding). As soon as at least $k \times (1.05)$ symbols are received, FEC decoding is launched. Otherwise, when a packet belonging to the

next source block arrives, a decision is taken: if at least k symbols are available, decoding is launched.

3.4 Performance Evaluation

3.4.1 Experimental Setup and Performance Metrics

To evaluate the performance of DVTS with FEC scheme, we set up an experimental environment. The sender and the receiver are desktop machines under 32bit Linux operating system, running a modified DVTS that includes the FECFRAME framework and several FEC schemes. The sender uses a Pentium 4/2.4 GHz CPU/512 MB RAM and runs a Linux 32bit OS. The receiver uses a Pentium Core 2/1.66 GHz CPU/1 GB RAM and runs a Linux 32bit OS. The DVTS application is version hdvts1.0a [74]. The 2D codes, Reed-Solomon codes, and LDPC-staircase codes are those available in version 1.0.0 of the `OpenFEC.org` library [28].

The sending strategy consists in sending all FEC source packets first, then FEC repair packets. Additionally, in case of LDPC-staircase codes, the repair packets are sent in a random order, whereas for the two other codes, they are sent sequentially. During tests the uniform packet loss probability used is progressively increased from 0% to 51%, by 3% each time. The packets erased are randomly chosen, in accordance with this loss probability. During each test (that last 60 seconds each), the receiver then measures the recovery capabilities, the frame delays and the CPU resource usage. The CPU resource usage is also measured at the sender side.

Table 3.1 shows the parameters for each FEC code considered. The code rate is fixed and set equal to 2/3 for all AL-FEC codes. $k = 170$ is chosen since it corresponds to $n = 255$, i.e. a value compatible with RSE over GF(2^8). Since we use DV format and the maximum DV/RTP packet size is 1372 bytes, we set $E =$ 1372+3 bytes. This has to be compared to the maximum packet loss probability set deliberately to 51%. We can expect that as the packet loss probability approaches 33.3%, performance will degrade, and above this value and until 51%, performance will be rather poor, a full decoding of a source block being usually impossible (but partial decoding may happen, especially with the "small" 2D codes).

Table 3.1: **FEC parameters.**

FEC Codes	Symbol Length (bytes)	Code Rate	Source Block Length (k)	N_1
2D Codes	1375	2/3	16	none
RSE Codes	1375	2/3	170	none
LDPC Codes	1375	2/3	(1) 170	7
			(2) 500	5
			(3) 1000	5

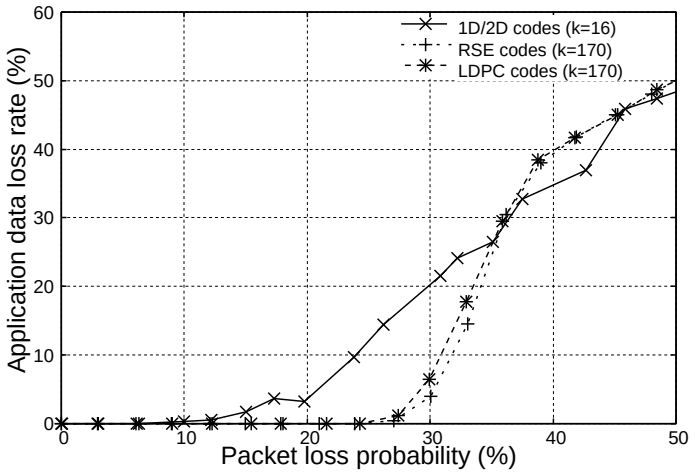


Figure 3.2: **Average data loss rate with 2D, RSE, and LDPC ($k = 170$) codes.**

3.4.2 Recovery Capabilities

The packet loss recovery capabilities can be evaluated by measuring the actual data loss percentage within the application (i.e. the DV/RTP packet loss percentage after loss correction within FECFRAME) as a function of the packet loss probability. Fig. 3.2 and Fig. 3.3 show that there remain unrecovered loss with a 30% packet loss probability (or higher). With 2D codes the situation is worse since about 0.03% unrecovered data losses happen even with a packet loss probability lower than 10%. Above 10%, unrecovered loss quickly increases. With RSE and LDPC codes, the unrecovered loss is null below approximately a 30% of packet loss probability. We see that RSE codes ($k = 170$) perform only marginally better than LDPC ($k = 170$) codes. Increasing LDPC's k parameter to $k = 500, 1000$

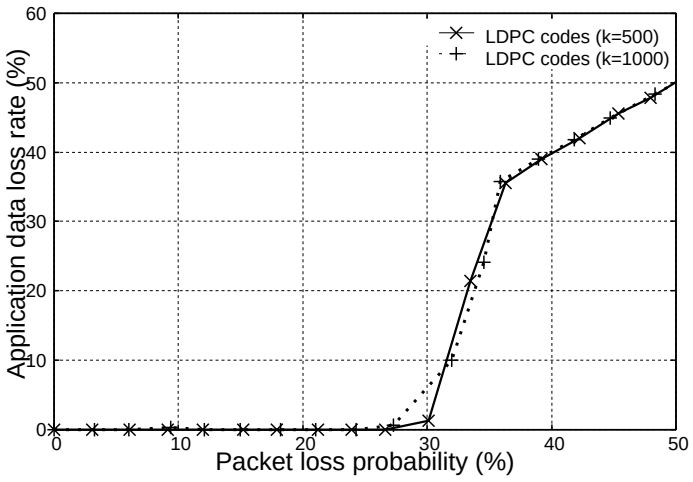


Figure 3.3: **Average data loss rate with LDPC codes ($k = 500, 1000$).**

(we know that LDPC codes perform asymptotically well) improves marginally the erasure recovery capabilities (while increases the decoding delay which is clearly an issue).

3.4.3 Frame Delay

We then evaluate the frame delay, during receiver's RTP processing, by measuring how long it take to the FECFRAME instance to get each DV/RTP packet, compared to a normal DVTS processing. To that purpose, we measure the number of delayed frames and their delay. The average frame delay in each test is then obtained by dividing the total delay time by the number of delayed frames. This average frame delay, plus the standard deviation above and below, as a function of the packet loss probability is shown in Fig. 3.4 and Fig. 3.5. With 2D codes, throughout the tests (from 0% to 51% loss probabilities), the maximum average frame delay turned out to be 63.7 ms (around a 26.2% packet loss probability). Otherwise, the average frame delay was usually lower than 50 ms.

With RSE codes, the average frame delay around a 12.1% packet loss probability was over 100 ms. Then around 30.1% packet loss probability, the average frame delay became 543.9 ms which is rather bad.

With LDPC codes ($k = 170$), when the packet loss probability was lower

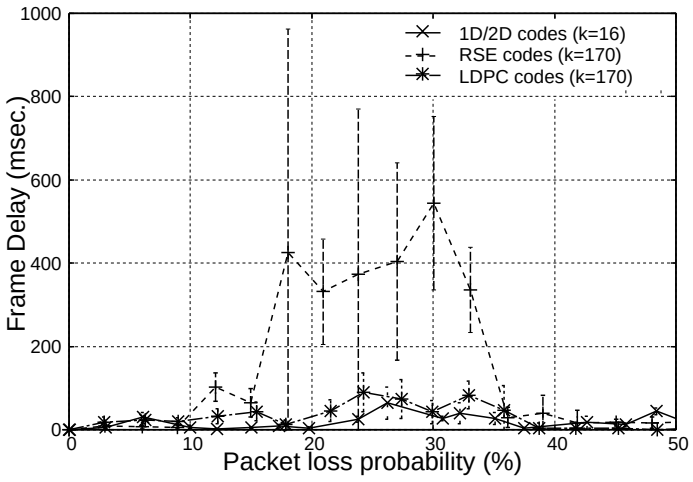


Figure 3.4: **Frame delay with 2D, RSE, and LDPC ($k = 170$) codes.**

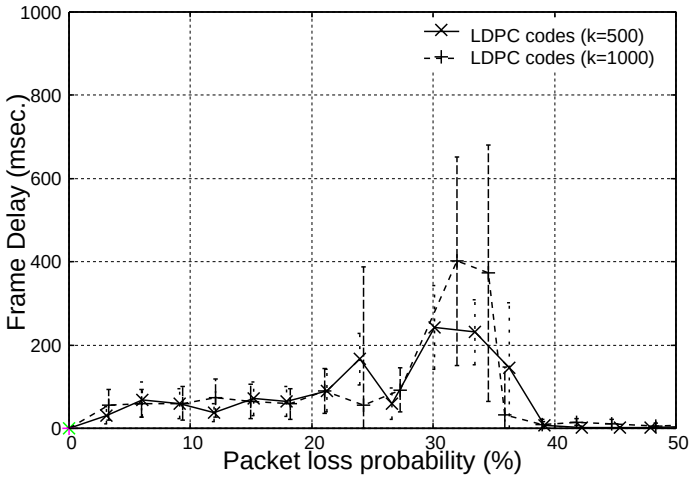


Figure 3.5: **Frame delay with LDPC codes ($k = 500, 1000$).**

than 24%, the average frame delay was below 50 ms. Above 24% packet loss probability, the maximum average frame delay was about 89.4 ms. With larger LDPC codes ($k = 500, 1000$), the average frame delay was always greater than that of LDPC codes ($k = 170$). In particular, in high packet loss probability conditions (more than 24%), the average frame delay of LDPC codes with the large source block length was significantly greater. For instance, around a 30% packet loss probability, this average delay was about 241.6 ms ($k = 500$) and 401.5

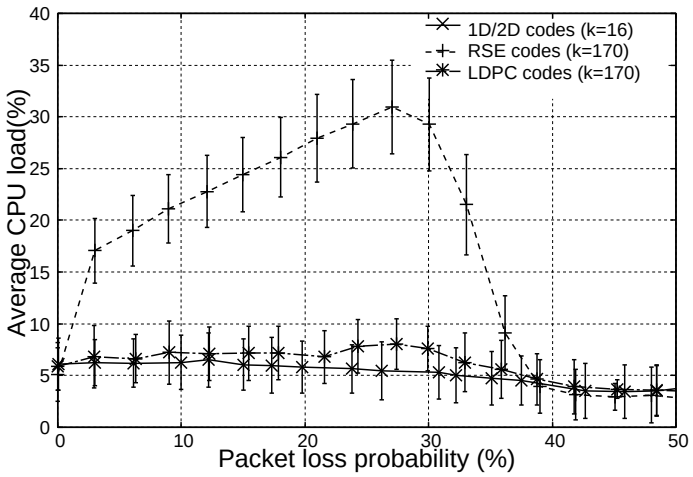


Figure 3.6: **CPU load at the receiver side, with 2D, RSE, and LDPC ($k = 170$) codes.**

ms ($k = 1000$), respectively.

From these results we can say that in our use-case, small LDPC codes ($k = 170$) are a good solution since they offer good erasure capabilities (close to that of Reed-Solomon codes of similar dimension) while keeping the frame delay low, close to that of 2D codes.

3.4.4 Processing Load

In order to understand resource usage at the receiver side during FECFRAME processing, we measured the CPU load at one second interval. Fig. 3.6 and Fig. 3.7 show the average CPU load at the receiver, plus the standard deviation above and below, as a function of the packet loss probability. As seen from Fig. 3.6, the average CPU load of RSE codes quickly increases until the packet loss probability reaches approximately 30%. For instance, at 27.1% packet loss probability, the average CPU load at the receiver is 31.0%. For higher packet loss probabilities, this load decreases since the receiver has fewer and fewer opportunities to launch RSE decoding. On the opposite, with the 2D and LDPC-staircase codes, no matter the packet loss probability over the network, the average CPU load always remains below 8%.

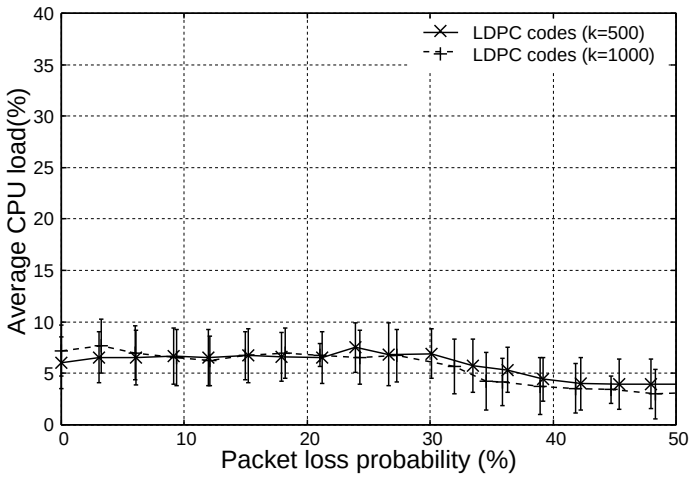


Figure 3.7: **CPU load at the receiver side, with LDPC codes** ($k = 500, 1000$)**.**

Table 3.2: **CPU load at the sender side.**

	2D codes	RSE codes	LDPC codes		
k, code rate	16, 2/3	170, 2/3	170, 2/3	500, 2/3	1000, 2/3
ave. CPU load (%)	14.1	74.3	12.0	11.0	12.6
std deviation	6.0	9.0	6.3	6.7	6.2

We also measured the CPU load at the sender side during FECFRAME processing. Table 3.2 shows the average CPU load and the corresponding standard deviation. We see that RSE encoding create a significant CPU load, compared to 2D and LDPC-staircase codes.

3.4.5 Discussions

Based on the experimental results in terms of recovery capabilities, frame delay, and processing load, we can identify the optimal configuration that achieves the best performance. In presence of a certain packet loss probability, it is necessary to find a well-balanced point between the recovery capabilities and frame delay. In general, in order to absorb the arrival delay of the source packet, a receiver requires a packet buffer of a certain length at the cost of the real-time performance. If the

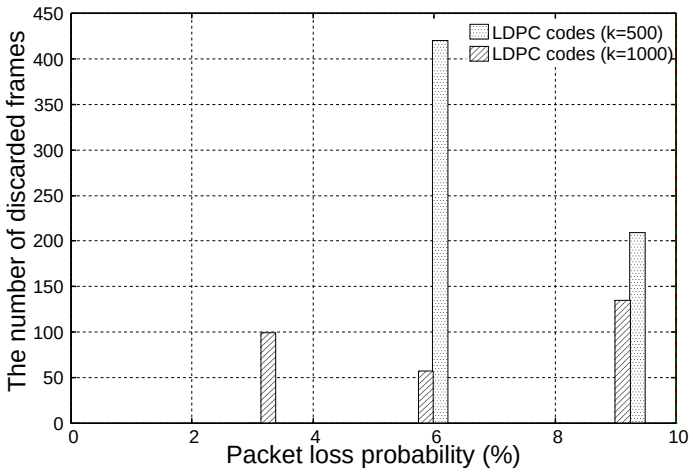


Figure 3.8: **The number of discarded frames when using the equivalent buffer of 100 ms, in the presence of the packet loss probability from 0% to 10%.**

packet arrival delay exceeds the acceptable range determined by the buffer length, the packet is discarded. Therefore, an appropriate tuning of this buffer length is vital.

Within less than 10% of packet loss probability, RSE and LDPC-staircase codes recover almost all the lost source packets. The 2D codes continually causes about 0.03% data loss rate because of the small source block length ($k = 16$). In the case of 2D, RSE, and LDPC ($k = 170$) codes, the receiver needs the equivalent buffer length of 100 ms in order to absorb the generated frame delay. However Fig. 3.8 shows that LDPC codes with larger source block lengths ($k = 500, 1000$) causes some frames to be discarded with the equivalent 100 ms buffer. With an equivalent 200 ms buffer (not shown), there is no discarded frame any more. In this context we can conclude that RSE and LDPC-staircase ($k = 170$) codes are suitable in the presence of low and non-burst packet loss conditions, in terms of both video quality and real-time performance. With a low spec machine, LDPC-staircase ($k = 170$) and 2D codes should be preferred because of the modest processing load generated. In presence of more than 10% packet loss probability, 2D codes cannot recover lost source packets, while RSE and LDPC-staircase codes fully recover them up to around 30% packet loss probability. As seen from Fig. 3.9, when using an equivalent 100 ms buffer, the number of discarded frames in all cases increases

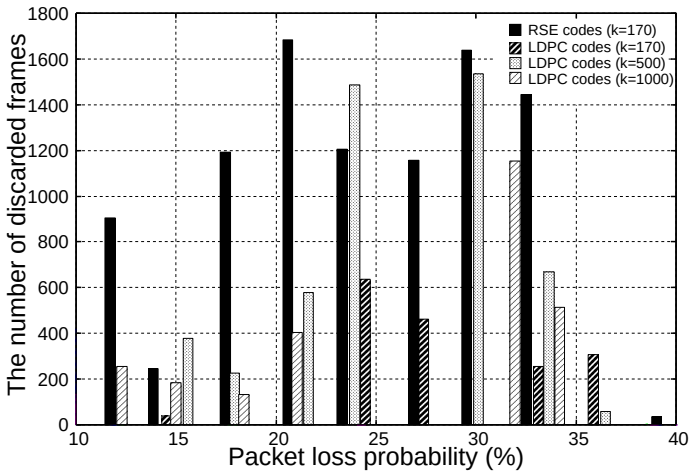


Figure 3.9: **The number of discarded frames when using the equivalent buffer of 100 ms, in the presence of the packet loss probability from 10% to 40%.**

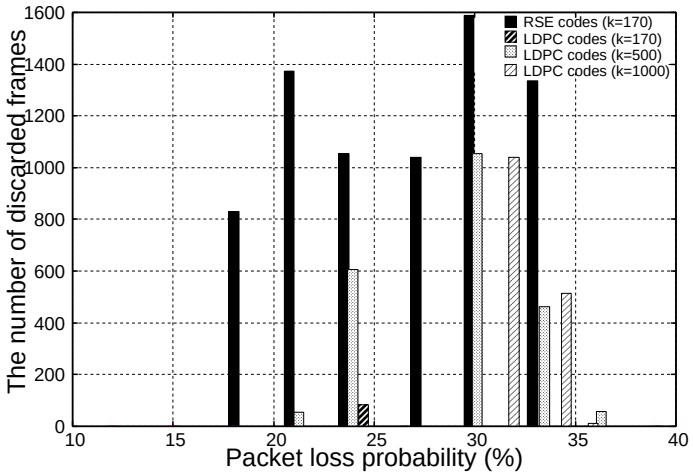


Figure 3.10: **The number of discarded frames when using the equivalent buffer of 200 ms, in the presence of the packet loss probability from 10% to 40%.**

as the packet loss probability increases. This is particularly true for RSE codes that behave badly. On the opposite, LDPC-staircase ($k = 170$) codes are the codes that perform the best (not considering 2D codes). In Fig. 3.10, with an equivalent 200 ms buffer, the number of discarded frames considerably decreases except for

RSE codes that still show poor performance. To a lesser extent, with high packet loss probabilities, LDPC codes ($k = 500, 1000$) feature a large number of discarded frames because of their important decoding time (more packets are needed before decoding can start). However, if a receiver can afford a large equivalent buffer length, at the cost of real-time performance, LDPC-staircase codes with large block lengths can achieve the higher recovery capability in high and bursty packet loss conditions which can be a significant asset.

To conclude, LDPC-staircase ($k = 170$) codes are good choices in terms of recovery capabilities, real-time performance and processing load, no matter the incoming packet loss rate.

3.5 Summary

In this chapter, we analyzed the benefits of adding three AL-FEC codes, namely the 2D parity check, Reed-Solomon (RSE) over GF(2^8) and LDPC-staircase codes, in a high-performance, high throughput real-time video streaming application, the Digital Video Transport System (DVTS). More precisely DVTS has been modified, following the FECFRAME approach [77] to integrate AL-FEC codes. Based on the experimental results in terms of recovery capabilities, frame delay and processing load, we have identified the optimal configuration for a given incoming loss rate. More precisely, under low packet loss conditions, all FEC codes achieve high quality video playback since erasures are recovered within their real-time constraints. However, from the processing load point of view, RSE codes create an important CPU load compared to the other codes. Under high packet loss conditions, the 2D and RSE codes no longer perform well in terms of the recovery capability and real-time performance. On the opposite, the LDPC-Staircase ($k = 170$) codes enable DVTS to achieve almost optimal performance. In the next chapter, we propose Dynamic Probing FEC (DP-FEC) mechanism, describe the DP-FEC design and algorithm, and evaluate the effectiveness of DP-FEC.

Chapter 4

DP-FEC: Dynamic Probing FEC

In this chapter, we propose the dynamic probing forward error correction (DP-FEC) [60] that can be utilized to maintain the best possible streaming quality for the purposes of IPTV, e-learning, or international symposiums. DP-FEC mechanism is effective for high-quality real-time streaming to maximize the streaming quality in a situation in which competing TCP flows pose packet losses to the streaming flow. DP-FEC estimates the network condition by dynamically adjusting the degree of FEC redundancy while trying to recover lost data packets. It effectively utilizes network resources and adjusts the degree of FEC redundancy to improve the playback quality at the user side while minimizing the performance impact of competing TCP flows. We describe the DP-FEC algorithm and evaluate its effectiveness using an NS-2 simulator. The results show that by effectively utilizing network resources, DP-FEC enables to retain higher streaming quality while minimizing the adverse condition on TCP performance, thus achieving TCP friendliness.

4.1 Design Overview

In this section, we propose the dynamic probing forward error correction (DP-FEC) mechanism to satisfy the requirements aforementioned. According to the network condition, the mechanism changes "FEC window size" (packets) to make a packet loss tolerance, while considering the impact of FEC on performance of

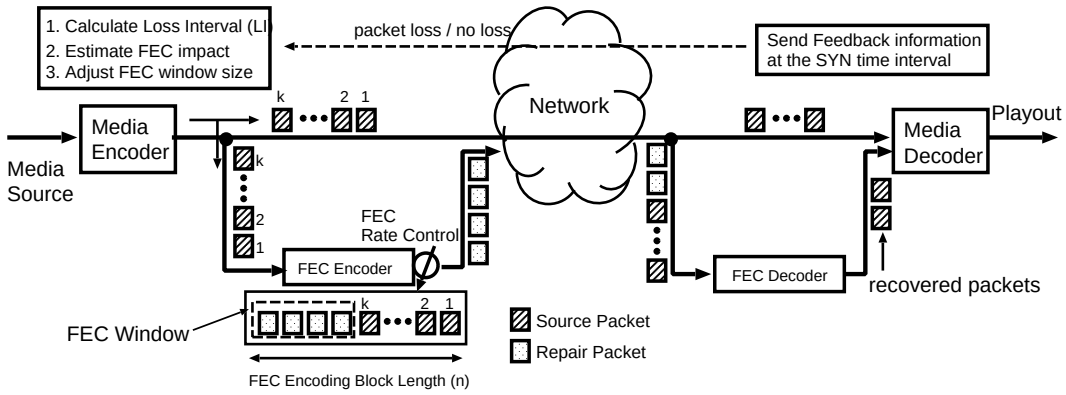


Figure 4.1: **The DP-FEC overview.**

competing TCP flows.

Fig. 4.1 illustrates the DP-FEC overview. DP-FEC operates mainly at the sender. A sender transmits data and FEC repair packets over a real-time transport protocol (RTP) [6] carried on top of the Internet Protocol (IP) and UDP. The DP-FEC collects the feedback information transmitted over the real-time transport control protocol (RTCP) delivered by a receiver, then adjusts the degree of FEC redundancy using feedback information about packet losses. The data transmission rate, which depends on the video format preliminarily assigned, is maintained during transmission.

DP-FEC leverages an application level forward error correction (AL-FEC). In AL-FEC, $n-k$ repair packets are added to a block of k data packets. We consider maximum distance separable (MDS) codes such as Reed-Solomon codes [7] which can recover all of the missing packets from any set of exactly k packets. Here, we define the number of repair packets as the "FEC window size", indicated by "$n-k$". If the code parameters, the FEC window size and n, are appropriately set in the event of packet losses, a receiver may recover all of the missing data packets within a block. DP-FEC changes the FEC window size in a fixed FEC encoding block length n during transmission. Furthermore, all of the generated data packets are stored once in the FEC encoding block buffer. Depending on the FEC window size, the number of data packets stored in the FEC encoding block buffer will vary.

DP-FEC is designed as an end-to-end model in consideration of the following

criteria:

1. Utilizing network resources effectively, the FEC window size during data transmission is increased to make a packet loss tolerance as high as possible. At the same time, network estimation for congestion control can be conducted.

2. In accordance with the network estimation, DP-FEC adjusts the FEC window size to minimize the impact of FEC on performance of competing TCP flows.

4.2 Algorithm

The DP-FEC algorithm consists of three components: 1) the congestion estimation function, 2) repeating congestion estimation, and 3) the FEC window size adjustment function. Next, we describe the algorithm in detail and the parameters used.

4.2.1 Congestion Estimation Function

The decision to adjust the FEC window size can be summarized as follows:

- If no congestion is sensed, the FEC window size is increased except when the FEC window size is already at its maximum.

- If congestion is sensed, the FEC window size is decreased except when the FEC window size is at its minimum.

The DP-FEC recognizes above a certain degree of FEC impact on competing TCP performance as an indicator of network congestion. This means that the DP-FEC does not just use packet loss as a congestion indicator, but allows a certain level of packet loss during transmission. Here, we assume that a DP-FEC flow competes with TCP flows in the same bottleneck link, and define FEC impact as a ratio of FEC_bw to TCP_abw; FEC_bw is the consumption bandwidth of FEC repair packets that a DP-FEC flow adds per unit time, and TCP_abw is the available

bandwidth that existing TCP flows can utilize per unit time. Thus, in competition with N TCP flows, the change in the degree of FEC impact (ΔFEC_impact) caused by added ΔFEC_bw is given by:

$$\begin{aligned} \Delta FEC\_impact &= (\Delta FEC\_bw/TCP\_abw) \\ &= (\Delta FEC\_bw/N)/(TCP\_abw/N) \\ &= (\Delta FEC\_bw/N)/(TCP1\_bw) \end{aligned} \tag{4.1}$$

where $TCP1\_bw$ is the data bandwidth that one TCP flow consumes. According to $TCP1\_bw$ and N, the FEC impact on the TCP performances varies. $TCP1\_bw$ can be estimated as follows [31]:

$$TCP1\_bw = \frac{PacketSize}{RTT\sqrt{\frac{2p}{3}} + RTO\left[3\sqrt{\frac{3p}{8}}p(1+32p^2)\right]} \tag{4.2}$$

where $PacketSize$ is the size of sending packet (bytes); p is the loss event rate; RTT is the Round Trip Time (sec); RTO is the retransmission timeout value (sec). If we can know N, ΔFEC_impact can be estimated. However, N changes from moment to moment, and it is naturally hard for an end system on an end-to-end model to precisely know N at the right time.

DP-FEC approximates $\Delta FEC\_bw/N$ by successively changing FEC_bw and observing the loss interval (LI) defined as the interval of time between packet loss events. An occurrence of more than one packet loss within a specific time is recognized as a single packet loss event. In order to approximate $\Delta FEC\_bw/N$, we now focus on the steady-state behavior of one TCP flow adopting the well-deployed AIMD algorithm [70] known as a mechanism for the remarkable stability of the Internet [32]. We assume that when a TCP flow observes more than one packet loss event during an RTT that results in halving the congestion window size, a DP-FEC flow also observes the same packet loss event. Since the TCP flow adopts the AIMD algorithm, LI can be approximated as follows:

$$LI \approx \frac{(TCP\_abw/N) \times RTT^2}{2 \times MSS} \tag{4.3}$$

where MSS is the TCP maximum segment size (bytes). When DP-FEC adds ΔFEC_bw, ΔLI can be approximated using Eq.(4.3), as follows:

$$\begin{aligned}\Delta LI &\approx \frac{(TCP\_abw/N) \times RTT^2}{2 \times MSS} \\ &- \frac{\{(TCP\_abw - \Delta FEC\_bw)/N\} \times RTT^2}{2 \times MSS} \\ &= (\Delta FEC\_bw/N) \times \frac{RTT^2}{2 \times MSS} \end{aligned} \tag{4.4}$$

Using Eqs.(4.1), (4.2) and (4.4), we can approximate ΔFEC_impact as follows:

$$\Delta FEC\_impact \approx \frac{\Delta LI}{TCP1\_bw} \times \frac{2 \times MSS}{RTT^2} \tag{4.5}$$

DP-FEC estimates ΔFEC_impact using an observed ΔLI. An observation of a larger value of ΔLI implies that the increasing FEC window size has a more negative impact on TCP performance by reducing TCP_abw. DP-FEC recognizes network congestion when an estimated ΔFEC_impact exceeds *Threshold FEC Impact*. The value of *Threshold FEC Impact* in DP-FEC is set to 0.1. Since an observed ΔLI is influenced by packet loss patterns that vary according to ever-changing network conditions (e.g., the number of competing TCP flows and the available bandwidth), precisely estimating ΔFEC_impact is naturally difficult. Therefore, by successively observing ΔLI while changing the FEC window size, DP-FEC evaluates whether ΔFEC_impact exceeds *Threshold FEC Impact* or not. Although this strategy leads to low responsiveness to a change in network conditions, DP-FEC can effectively utilize network resources by adding FEC repair packets while considering the impact of FEC on TCP flows. The decision frequency is described in next Section 4.2.2

As Eq.(4.5) shows, the estimation of ΔFEC_impact depends largely on RTT that competing TCP flows have, since TCP performance generally relies on RTT. If the RTT value is large, DP-FEC tends to conservatively behave and keep a small FEC window size even in slightly congested networks. This situation causes a number of non-recovered data packets without effectively utilizing network resources. To avoid such a behavior, DP-FEC estimates ΔFEC_impact assuming that all competing TCP flows have $RTT = 0.01$ (sec). Thus, in competition with

TCP flows with $RTT > 0.01$, the DP-FEC tends to keep a larger FEC window size due to the underestimation of ΔFEC_impact.

4.2.2 Repeating Congestion Estimation

A DP-FEC receiver sends feedback information at constant time interval (SYN time), which denotes whether packet loss happened or not during the SYN time. Multiple packet losses in the same SYN time are considered as a single loss event. As described before, DP-FEC assumes that 1) both a DP-FEC flow and TCP flow observe the same packet loss event during an RTT and 2) all competing TCP flows have $RTT = 0.01$ (sec). Thus, the value of SYN time in DP-FEC is set to 0.01 (sec). A sender receives feedback information at the SYN time interval, and calculates the sample loss interval ($SampleLI$), the minimum value of which is equivalent to 0.01 (sec). Using the Exponential Weighted Moving Average (EWMA), current LI is calculated as follows: $LI = \alpha \times LI + (1 - \alpha) \times SampleLI$. DP-FEC uses the smoothing factor $\alpha = 0.9$, and estimates ΔFEC_impact with the observed variation of LI (ΔLI) at the constant interval ($ResTime$). The $ResTime$ relates to the tradeoff between the aggressiveness (i.e., increasing rate of FEC window size) and estimation accuracy of FEC_impact (i.e., TCP-friendliness). In an ever-changing network condition, the appropriate setting of $ResTime$ value is quite difficult. In this dissertation, we set $ResTime$ to 0.16 (sec) based on our comprehensive simulation results as of now.

Fig. 4.2 shows the relation between the current LI and maximum acceptable ΔLI, where MSS=1460 (bytes), RTT=0.01 (sec). If an observed ΔLI exceeds the maximum acceptable ΔLI, a sender recognizes an occurrence of network congestion (i.e., $\Delta FEC\_impact > 0.1$). Note that if a network condition drastically becomes worse (e.g., due to an occurrence of bursty traffic), LI severely decreases. If this situation continues, DP-FEC needs to decrease the FEC window size in order to avoid congestion collapse. Therefore, DP-FEC also recognizes a network congestion when LI becomes less than the minimum value ($MinLI$). Every time a sender receives feedback information, the current LI is calculated and judged to be below $MinLI$. When DP-FEC uses an extremely-low value of $MinLI$, the attainable FEC window size becomes large even in highly congested networks due to the slow response to the network congestion. Such a situation causes a significant

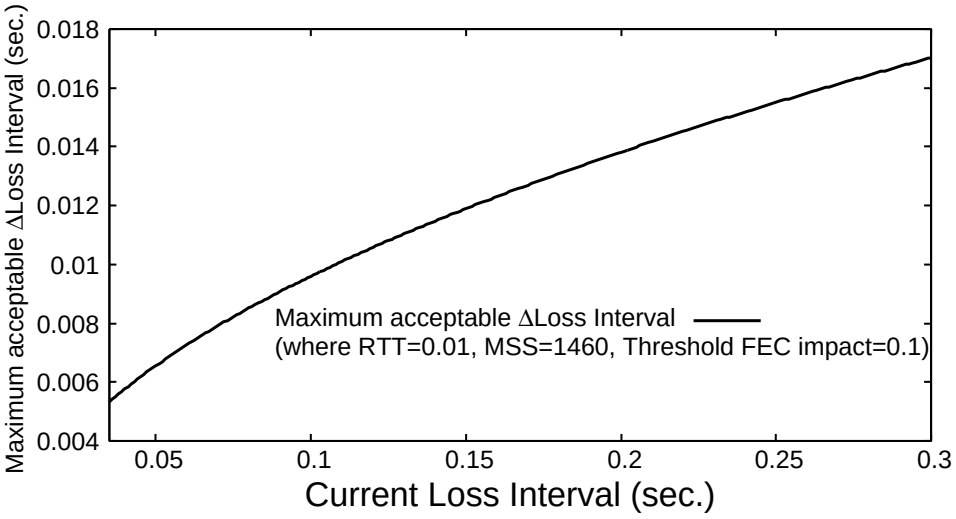


Figure 4.2: **Relation between the current calculated loss interval (LI) and maximum acceptable variance of loss interval (ΔLI).**

performance degradation of competing TCP flows. To avoid it, we set MinLI to 0.035 (sec) based on our simulation experiments as of now.

4.2.3 FEC Window Size Adjustment Function

DP-FEC recognizes network congestion when estimated ΔFEC_impact exceeds *Threshold FEC Impact* or current LI falls below $MinLI$. In DP-FEC, AIMD algorithm is used to adjust the FEC window size. FEC window size ($Fwnd(t)$) in packets is given by:

$$Fwnd(t) = \begin{cases} Fwnd(t-1) + 1 & per\ ResTime \\ \lfloor \beta Fwnd(t-1) \rfloor & per\ congestion \end{cases} \tag{4.6}$$

During no congestion, $Fwnd$ is linearly increased. During congestion, DP-FEC immediately decreases $Fwnd$ using multiplicative factor β. DP-FEC with a low value of β tends to fail to recover lost data packets in continuously congested networks, because the attainable FEC window size becomes small. Since 1) DP-FEC should try to recover data loss as much as possible, and 2) by estimating and controlling ΔFEC_impact, DP-FEC can avoid a negative impact on TCP

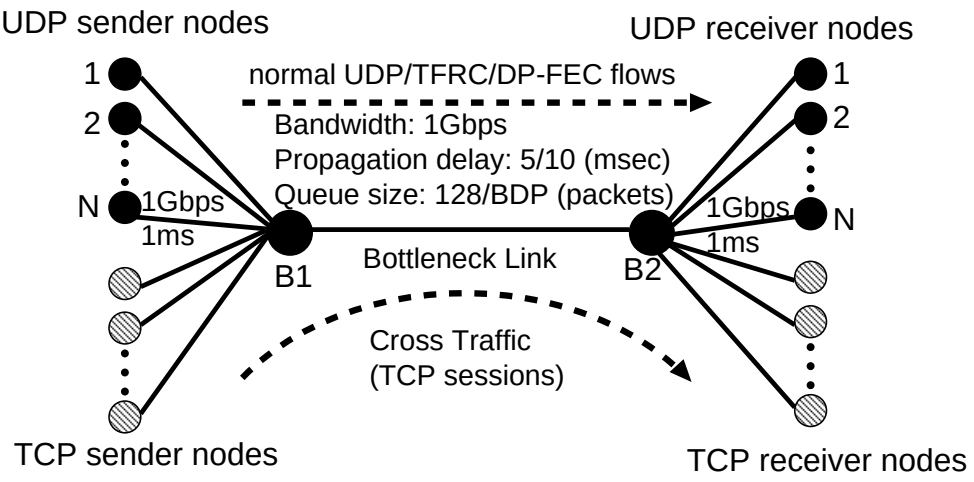


Figure 4.3: **Simulation network model.**

performance in network conditions where added FEC redundancy significantly disturbs TCP performances, DP-FEC sets β to a relatively high value, 0.8 (i.e., more than 0.5).

4.3 Evaluation

4.3.1 Simulation Setup and Performance Metrics

Using an NS-2 simulator extended with DP-FEC module, we performed experiments using the dumbbell topology shown in Fig. 4.3. The bandwidth of the bottleneck link between $B1$ and $B2$ was set to 1 Gbps. The propagation delay was set to 5 or 10 ms. A drop-tail queuing was used at the router, and the queue length size in packets was set to 128 or BDP. Each sender and receiver were connected to the bottleneck link through the 1 Gbps access link with a propagation delay of 1 ms. The packet size was set to 1500 bytes for all connections. Each simulation ran for about 120 seconds.

In all simulation experiments, a DP-FEC sender takes the same algorithm for sending data packets. It transmits smoothed data packets at a rate of 30 Mbps. As a practical matter, both the FEC encoding block length and the maximum FEC

window size should be determined by the acceptable FEC encoding / decoding time for the streaming application. Here, we set the FEC encoding block length to 127 and the maximum FEC window size to 63, modeling a real environment.

As the DP-FEC performance metrics, we observed 1) the average residual data loss rate of DP-FEC flows (i.e., the rate of non-recovered data packet) and 2) the average throughput of competing TCP flows. The metric of the average residual data loss rate of DP-FEC flows was compared with those observed when we used a normal UDP flow at a rate of 30 Mbps or a TFRC flow [31] under the same network condition. As TCP friendliness index ($TFindex$), we use the result of the average throughput of TCP flows observed when they compete with TFRC flows, and define $TFindex$ as follows:

$$TFindex = \frac{TCPthuput\_Target}{TCPthuput\_TFRC} \tag{4.7}$$

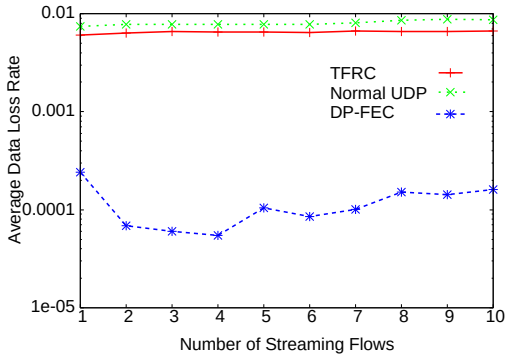
where $TCPthuput\_Target$ is the result of the average throughput of TCP flows competing with normal UDP flows or DP-FEC flows; $TCPthuput\_TFRC$ is the result of the average throughput of TCP flows competing with TFRC flows under the same network condition.

We used two types of TCP flows using a NewReno, 1) Short-lived TCP flows and 2) Long-lived TCP flows. Short-lived TCP flows arrive at the bottleneck link, as a Poisson process with an average rate of r_tcp flows per second. The size of each TCP flow follows a Pareto distribution with an average of s_tcp packets and shape parameter 1.5. Here, we define the load of short-lived TCP flows as $\rho\_tcp = r\_tcp \times s\_tcp$. Long-lived TCP flows are persistent in the network.

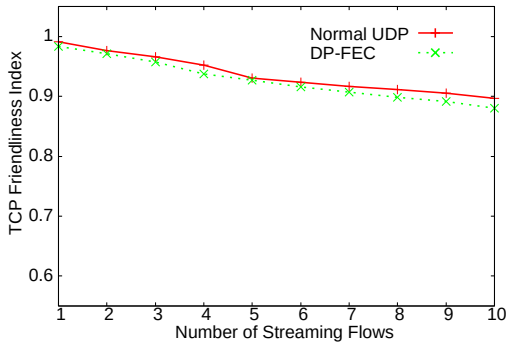
4.3.2 Competition with Only Short-Lived TCP Flows

We evaluated the DP-FEC performance in competition with only short-lived TCP flows. In this experiment, the propagation delay of the bottleneck link was set to 10 ms, and the queue size was set to 128 packets. The load of short-lived TCP flows (ρ_tcp) was set to 25%, 50% and 75% of the bottleneck link capacity. In addition, the number of UDP streaming flows (i.e., TFRC, normal UDP or DP-FEC flows) was changed from 1 to 10.

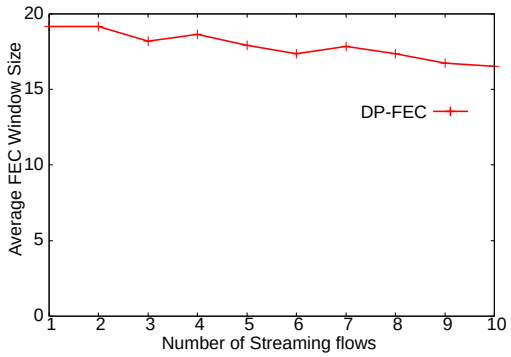
As shown in Fig. 4.4(a), the average data loss rates of DP-FEC flows under



(a) Average data loss rate



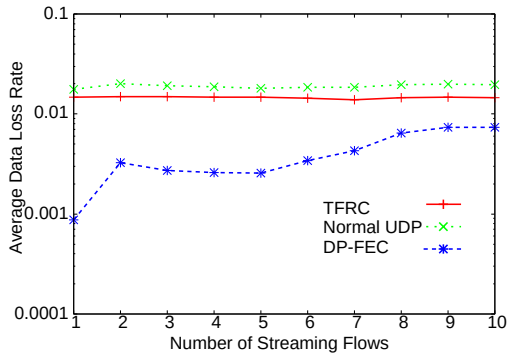
(b) TCP-friendliness index



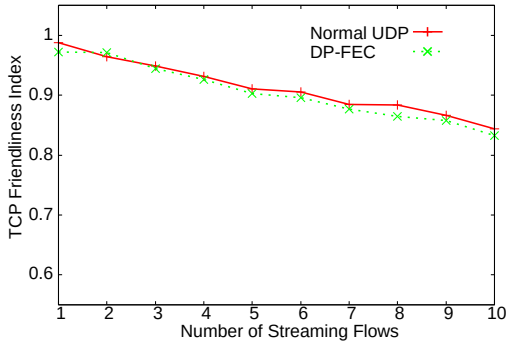
(c) Average FEC window size

Figure 4.4: **DP-FEC performances and average FEC window size under 25% load of TCP flows.**

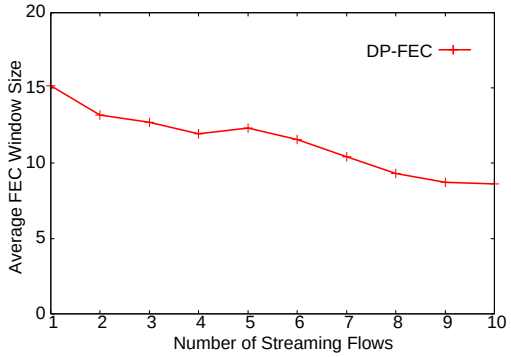
25% load of TCP flows are considerably improved by added FEC redundancy, compared to those of TFRC or normal UDP flows. In addition, Fig. 4.4(b) shows that the TCP friendliness indexes of both normal UDP and DP-FEC flows are maintained at more than approximately 0.9, which means that DP-FEC flows add a well-coordinated degree of FEC redundancy by effectively utilizing network resources that competing TCP flows cannot consume. In Fig. 4.4(c), we can see that the FEC window sizes are almost the same (between 17 and 20) regardless of the number of competing DP-FEC flows, because FEC impacts that each DP-FEC flow estimates during transmission are not largely different (i.e., there is little difference in the available bandwidth for TCP flows in each test). Because of an occurrence of packet losses caused by a behavior in slow start phase of generated short-lived TCP flow, the average FEC window size is suppressed to below 20. TFRC flows in each test also observe packet losses caused by a slow start phase of



(a) Average data loss rate



(b) TCP-friendliness index

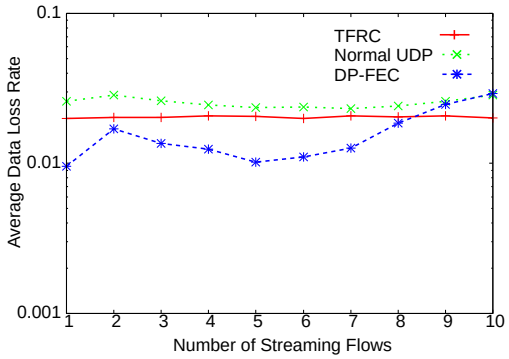


(c) Average FEC window size

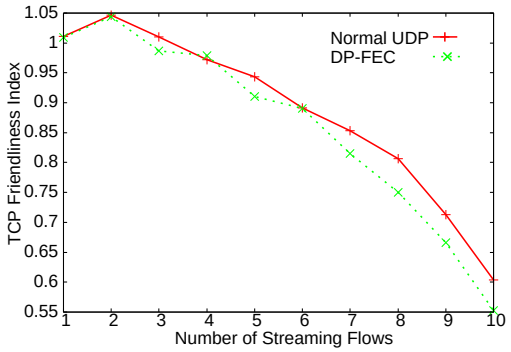
Figure 4.5: **DP-FEC performances and average FEC window size under 50% load of TCP flows.**

generated short-lived TCP flow, and reduce the data transmission rates to about 6 Mbps. As opposed to a TFRC flow, a DP-FEC flow suppresses the average data loss rate and preserves the highest data transmission rate while achieving high TCP friendliness index.

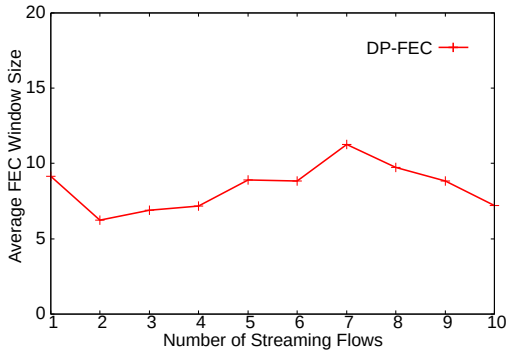
When ρ_tcp is 50% of the bottleneck link capacity, the average data loss rates of TFRC and normal UDP flows becomes more than 1%, as shown in Fig. 4.5(a). On the other hand, owing to added FEC redundancy, the average data loss rates of DP-FEC flows are suppressed to below 1%. In addition, Fig 4.5(b) shows that although the TCP friendliness indexes of both normal UDP and DP-FEC flows decrease with an increase in the number of streaming flows, DP-FEC flows in each test retains more than approximately 0.85% of the TCP friendliness index. In Fig. 4.5(c), we can see that as the number of DP-FEC flows increases, the average FEC window size gradually decreases to about 9. This is because, under 50%



(a) Average data loss rate



(b) TCP-friendliness index



(c) Average FEC window size

Figure 4.6: **DP-FEC performances and average FEC window size under 75% load of TCP flows.**

load of TCP flows, DP-FEC flows tend to experience continuous packet loss and estimate higher FEC impact with an increase in the number of DP-FEC flows. The continuous packet loss leads to an increase in average data packet loss rate, as shown in Fig 4.5(a).

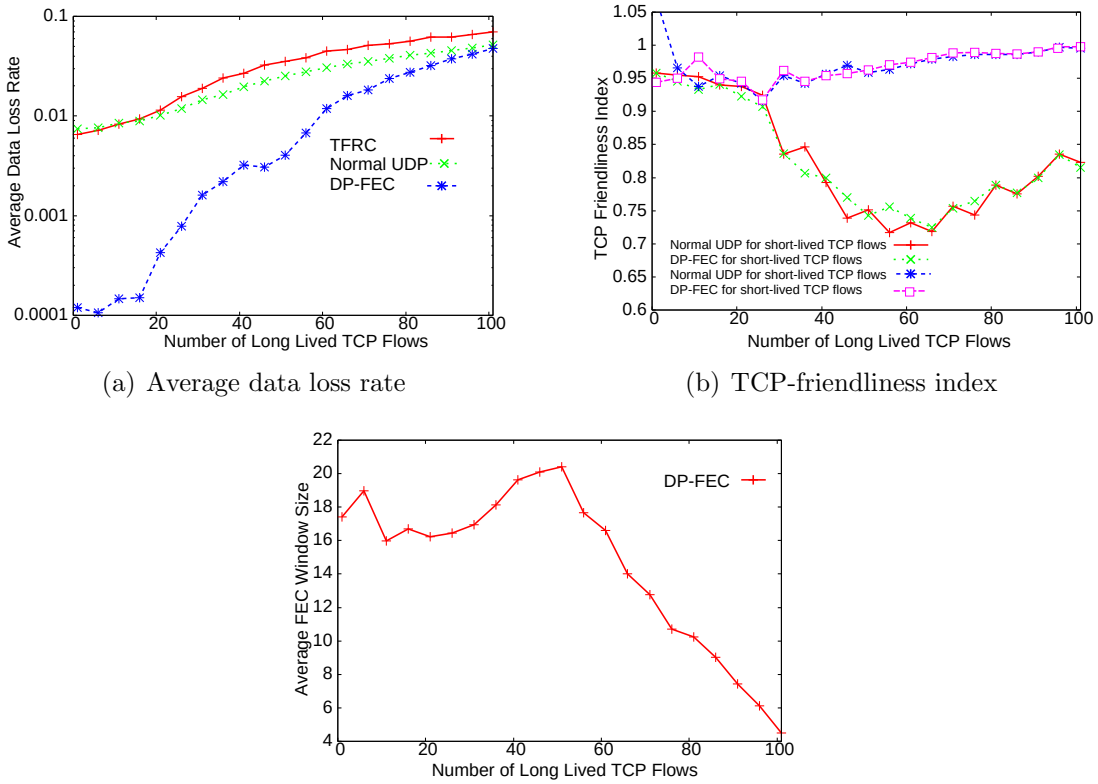
When ρ_tcp is 75% of the bottleneck link capacity, the average loss rates of DP-FEC flows are slightly improved except when the number of DP-FEC flows is more than 8, as shown in Fig. 4.6(a). In Fig. 4.6(b), we can see that as the number of streaming flows increases, the TCP friendliness indexes of both flows of Normal UDP and DP-FEC severely decrease because of a reduction in the available bandwidth for heavy short-lived TCP traffic. Due to the continuous packet loss caused by heavy short-lived TCP traffic, the average FEC window sizes are suppressed to between 5 and 10, as shown in Fig. 4.6(c). Since DP-

FEC is designed to minimize FEC impact defined as a ratio of the consumption bandwidth of FEC repair packets to the available bandwidth for TCP flows, the TCP friendliness indexes of DP-FEC flows become approximately the same as those of normal UDP. Therefore, in a situation in which a normal UDP flow severely decreases the TCP friendliness index, a DP-FEC cannot retain a higher TCP friendliness index.

4.3.3 Competition with Both Short-Lived TCP Flows and Long-Lived TCP Flows

We evaluated the DP-FEC performance in competition with both short-lived TCP flows and long-lived TCP flows. The parameters of the bottleneck link were the same as the previous experiments. The load of short-lived TCP flows (ρ_tcp) was set to 25% of the bottleneck link capacity, and the number of long-lived TCP flows was changed from 1 to 101. The number of UDP streaming flows was set to 3.

Fig 4.7(a) shows that whereas the average data loss rates of TFRC and normal UDP flows become more than 1% in competition with more than 20 of long-lived TCP flows, those of DP-FEC flows are suppressed to below 1% in competition with less than 60 of long-lived TCP flows. As the number of long-lived TCP flows increases, the average data loss rates of DP-FEC flows also increase and approach those of TFRC and normal UDP flows. This is because many long-lived TCP flows probe available bandwidth during the congestion avoidance phase and incur more continuous packet loss. Due to the continuous packet loss, DP-FEC flows recognize higher FEC impact, resulting in the reduction of average FEC window sizes, as shown in Fig. 4.7(c). However, the average FEC window sizes are kept between 15 and 20 when the number of long-lived TCP flows is less than or equal to 60. This condition is almost the same as in the case of competition with only short-lived TCP flows causing 25% load, because a DP-FEC flow recognizes that there is a certain level of available bandwidth by observing the interval between packet loss events while adjusting the FEC window size (i.e., the available bandwidth for TCP flows is not largely different despite increasing in FEC window size). Fig. 4.7(b) shows the TCP friendliness indexes of normal UDP flows and DP-FEC flows. The TCP friendliness index is distinguished by the type of TCP flow. Although DP-



(a) Average data loss rate

(b) TCP-friendliness index

(c) Average FEC window size

Figure 4.7: **DP-FEC performances and average FEC window size in competition with long-lived TCP flows, under 25% load of TCP flows.**

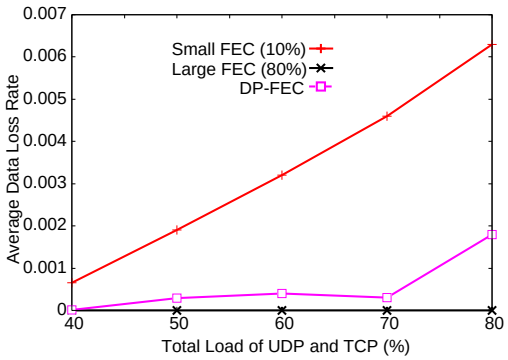
FEC flows retain the average FEC window size between 15 and 20 in competition with less than 60 of long-lived TCP flows, the TCP friendliness indexes for long-lived TCP flows always become more than 0.9. Whereas a TFRC flow reduces the data transmission rate to below about 6 Mbps under the influence of packet loss conditions, a DP-FEC flow utilizes network resources effectively by adding FEC repair packets which contribute to the recovery of data packet losses. On the other hand, the TCP friendliness indexes of both normal UDP and DP-FEC flows for short-lived TCP flows become more than 0.9 when the number of long-lived TCP flows is below 25. When the number of long-lived TCP flows becomes more than 25, the TCP friendliness indexes for short-lived TCP flows becomes between 0.7 and 0.9. This is because the total consumption bandwidth of long-lived TCP flows increases, which deteriorates the performances of short-lived TCP flows. However,

as described in Section 4.3.2, the TCP friendliness indexes of DP-FEC flows for short-lived TCP flows become approximately the same as those of normal UDP, since DP-FEC flows try to suppress the impact of FEC on the TCP performances.

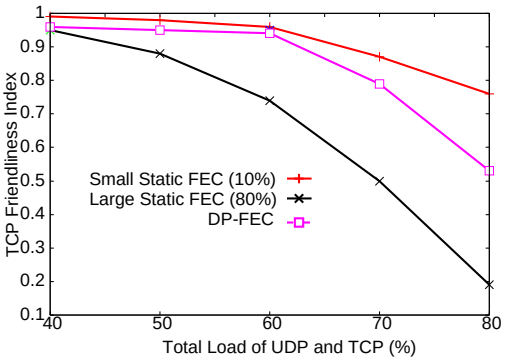
4.3.4 Comparison with Static-FEC Flow

We evaluated the DP-FEC performance to compare with that of small/large Static-FEC flows (i.e., the *Fwnd* is constant). The redundancy ratio (i.e., the ratio of repair packets to data packets in a block) of small Static-FEC flow was set to 10%, which is the same value with Pro-MPEG method [35]. The redundancy ratio of large Static-FEC flow was set to 80%, which is a fully large value. In this experiment, to investigate the DP-FEC performance when a TCP flow becomes more aggressive and unstable, the propagation delay of the bottleneck link was set 5 ms, and the queue size was set to a value equivalent to BDP (i.e., packets) [65], the value of which is much larger than 128 used in the previous experiments. The number of streaming flows is set to 10 in the all experiments.

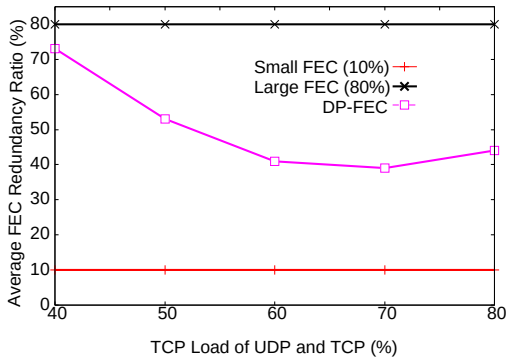
Fig. 4.8 shows the performance metrics and the average FEC redundancy ratio of DP-FEC flows when the total load of UDP (not including FEC) and TCP is between 40% and 80% of the bottleneck link capacity. We can see that, in Fig. 4.8(a), DP-FEC maintains much lower rates of residual data packet loss, compared to small Static-FEC flows. This is because DP-FEC flows aggressively increase the FEC window size and not easily reduce the size under moderate congestion as seen in Fig. 4.8(c). Nevertheless, DP-FEC maintains high TCP friendliness index (more than 90%) when the total load is between 40% and 60%. This means that DP-FEC effectively utilizes the available network resources by adding FEC redundancy. When the total load is 80%, the TCP friendliness index of DP-FEC flows becomes much lower than that of small Static-FEC flows. In such a situation in which many TCP flows compete with the DP-FEC flows and the network load always becomes high, the TCP friendliness index of DP-FEC severely decreases due to the high network load (i.e., 80% of bottleneck link capacity) and the aggressive FEC adding by DP-FEC flows. Since a severe degradation of TCP throughput tends not to cause packet loss event, DP-FEC flows cannot suppress adding FEC redundancy as shown in Fig. 4.8(c). On the other hand, large Static-FEC flows, which constantly add almost the same redundancy ratio with the maximum FEC



(a) Average data loss rate



(b) TCP friendliness index



(c) Average FEC redundancy ratio (%)

Figure 4.8: **DP-FEC performances when the total load of UDP flows (10 streaming flows without FEC) and TCP flows is between 40% and 80%.**

redundancy that DP-FEC can use, always suppress the average residual data loss rates to almost zero. However, when the total load becomes higher than 50%, the TCP friendliness indexes of large Static-FEC flows considerably degrades (up to about 0.2). DP-FEC flows in the total loads of more than 50% maintain higher TCP friendliness indexes than those of large Static-FEC flows while suppressing the residual data loss rates between about 0.0005 or less. In this context, compared with the small/large Static-FEC flows, DP-FEC effectively utilizes network resources to recover lost data packets while maintaining relatively higher TCP performance.

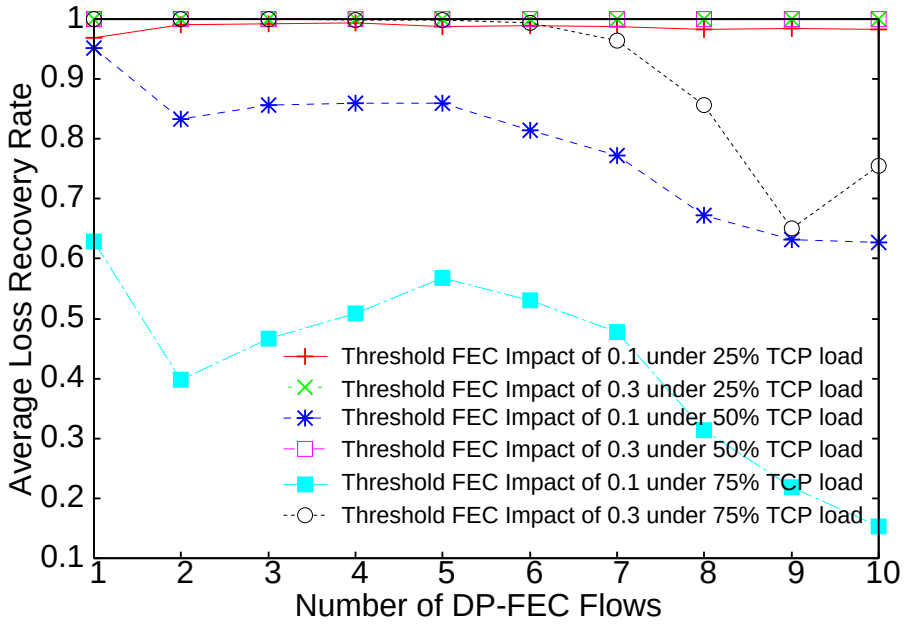


Figure 4.9: **Average loss recovery rate of DP-FEC flows (the cases of *Threshold FEC Impact* of 0.1/0.3 under TCP load of 25%/50%/75%).**

4.3.5 Effect of the Value of Threshold FEC Impact

The value of *Threshold FEC Impact* (the default value is 0.1) is considerably important for DP-FEC to recognize network congestion and adjust the FEC window size. Thus, we evaluated the performances of DP-FEC with *Threshold FEC Impact* of 0.3 under TCP load of 25%/50%/75%, and compared the results with those of DP-FEC with *Threshold FEC Impact* of the default value. The network environment is the same as that in the experiments of competition with only short-lived TCP flows.

Fig. 4.9 shows the average loss recovery rates of DP-FEC flows (the cases of *Threshold FEC Impact* of 0.1/0.3) under 25%/50%/75% load of TCP flows. The loss recovery rate is defined as the ratio of the number of the recovered data packets to that of the lost data packets in the network. We can see that under 25% load of TCP flows, DP-FEC flows with *Threshold FEC Impact* of 0.1 maintain the average loss recovery rates of more than 0.98. However, under both 50% and 75% load of TCP flows, the average loss recovery rates decrease because the attainable

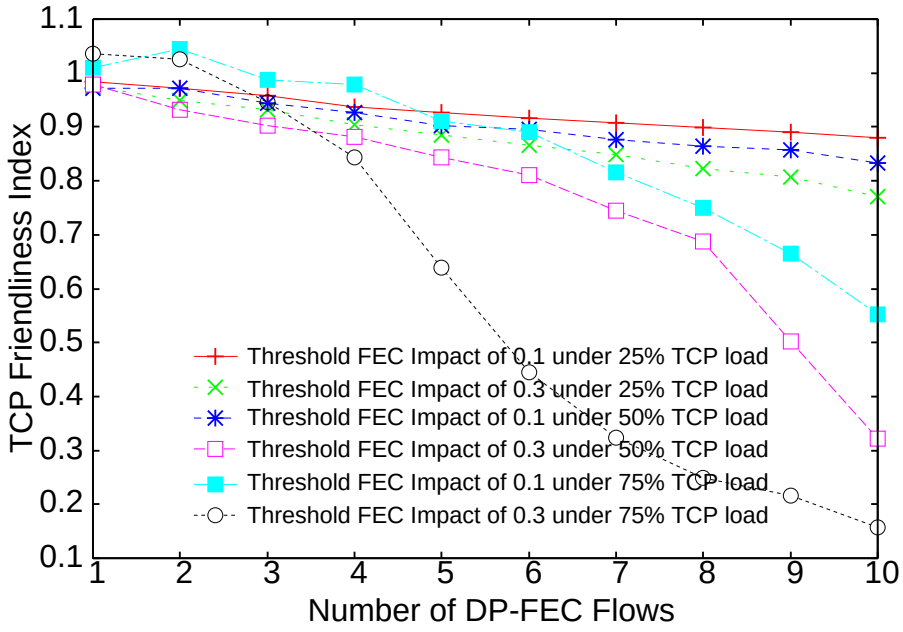


Figure 4.10: **TCP-friendliness indexes of DP-FEC flows (the cases of *Threshold FEC Impact* of 0.1/0.3 under TCP load of 25%/50%/75%).**

FEC window size of each DP-FEC flow tends to become small as described in Section 4.3.2. On the other hand, the average loss recovery rates of DP-FEC flows with *Threshold FEC Impact* of 0.3 almost become 100% except when the number of DP-FEC flows under 75% load of TCP flows is more than 6. This is because a DP-FEC flow with *Threshold FEC Impact* of 0.3 keeps larger FEC window size than that of a DP-FEC flow with *Threshold FEC Impact* of 0.1.

Fig. 4.10 shows the corresponding TCP friendliness indexes of DP-FEC flows. When the number of competing DP-FEC flows is the same, the TCP friendliness indexes of DP-FEC flows with *Threshold FEC Impact* of 0.3 becomes lower than those of DP-FEC flows with *Threshold FEC Impact* of 0.1. Under 25% load of TCP flows, the difference of TCP friendliness indexes is not large regardless of the number of DP-FEC flows. However, as the number of DP-FEC flows increases under 50% or 75% load of TCP flows, the difference becomes large (i.e., the TCP friendliness indexes of DP-FEC flows with *Threshold FEC Impact* of 0.3 largely decrease). This is because, since DP-FEC flows with *Threshold FEC Impact* of 0.3 tends to keep a larger FEC window size, the total bandwidth of FEC repair

packets increases considerably. Since the value of *Threshold FEC Impact* relates to the tradeoff between efficiency (i.e., loss recovery rate) and TCP friendliness, it is of greater importance for DP-FEC to appropriately set the value.

In this context, in a situation in which network load is relatively low, DP-FEC with the default value of *Threshold FEC Impact* achieves high loss recovery rate while maintaining high TCP friendliness indexes. However, in high network load, DP-FEC with the default value of *Threshold FEC Impact* causes lower loss recovery rate due to an emphasis on TCP friendliness index. When users can reflect an emphasis on high loss recovery rate rather than TCP friendliness index, the higher value of *Threshold FEC Impact* should be used.

4.4 Summary

In this chapter, we proposed the DP-FEC that effectively achieves streaming quality for a high-performance real-time streaming. This mechanism tolerates packet loss in the network through the adjustments of the FEC window size while examining network congestion. By successively observing variation in the intervals between packet loss events, DP-FEC effectively utilizes network resources and adjusts the degree of FEC redundancy while minimizing the impact of FEC on the performance of competing TCP flows. We verified the efficiency of DP-FEC using an NS-2 simulator, and recognized that DP-FEC retains high streaming quality while cooperating with TCP flows. In the next chapter, we describe the GENEVA design and algorithm, and evaluate the effectiveness of GENEVA.

Chapter 5

GENEVA: Real-time Streaming Control Algorithm using GMIAD

In this chapter, we propose GENEVA, the streaming control algorithm using generalized multiplicative-increase/additive-decrease (GMIAD) mechanism [61], assuming an usage environment where real-time streaming flows with various data transmission rates and RTTs compete (e.g., in the Internet). GENEVA avoids network underutilization by allowing a streaming flow to maintain moderate network congestion while trying to recover lost data packets that other competing flows cause during the process of probing for available bandwidth. Using the GMIAD mechanism, the FEC window size (the degree of FEC redundancy per unit time) is adjusted to suppress occurrences of bursty packet loss, while trying to effectively utilize network resources that other competing flows cannot consume due to reductions in the transmission rate in response to packet loss. We describe the GENEVA algorithm and evaluate its effectiveness using an NS-2 simulator. The results show that GENEVA enables high-performance streaming flows to retain higher streaming quality under stable conditions while minimizing an adverse impact on competing TCP performance.

5.1 Design Overview

GENEVA is designed as an end-to-end model using the following criteria:

1. To achieve high network utilization, GENEVA allows a high-performance streaming flow to maintain moderate network congestion, in which packet loss rate is less than p_{max} (a predefined constant value).

2. A sender with GENEVA needs to transmit data packets at an almost constant interval time, and control the degree of FEC redundancy to its data stream in congested networks. This situation may adversely increases traffic congestion. An GENEVA flow thus should converge at an appropriate equilibrium point to recover data packet losses under stable conditions during transmission. To achieve such a condition, supporting window control is needed to specify an acceptable degree of FEC redundancy.

3. To suppress an occurrence of bursty packet loss that TCP flows cause and improve FEC recover capabilities, the degree of FEC redundancy is increased to prevent their congestion window sizes from increasing considerably. At the same time, the increase in FEC redundancy should not interact badly with competing TCP flows.

In accordance with the aforementioned design overview (3), to improve high-performance streaming quality by FEC recovery of lost data packets, GENEVA should try to keep or increase the degree of FEC redundancy even in the presence of packet loss while utilizing expected available network resources which TCP flows cannot consume by their nature, and prevent TCP flows from achieving larger window size so as not to pose bursty packet loss. From this point of view, GENEVA does not adopt the well-deployed Additive-Increase/Multiplicative-Decrease (AIMD) algorithm [70] that decreases the transmission rate reacting to packet loss, but adopts GMIAD that increases the transmission rate reacting to packet loss (i.e., GMIAD increases the degree of FEC redundancy in GENEVA). Meanwhile, GENEVA using GMIAD decreases the degree of FEC redundancy during the period of no packet loss, where unnecessary FEC redundancy is suppressed so as not to trigger further network congestion in accordance with the design overview (2).

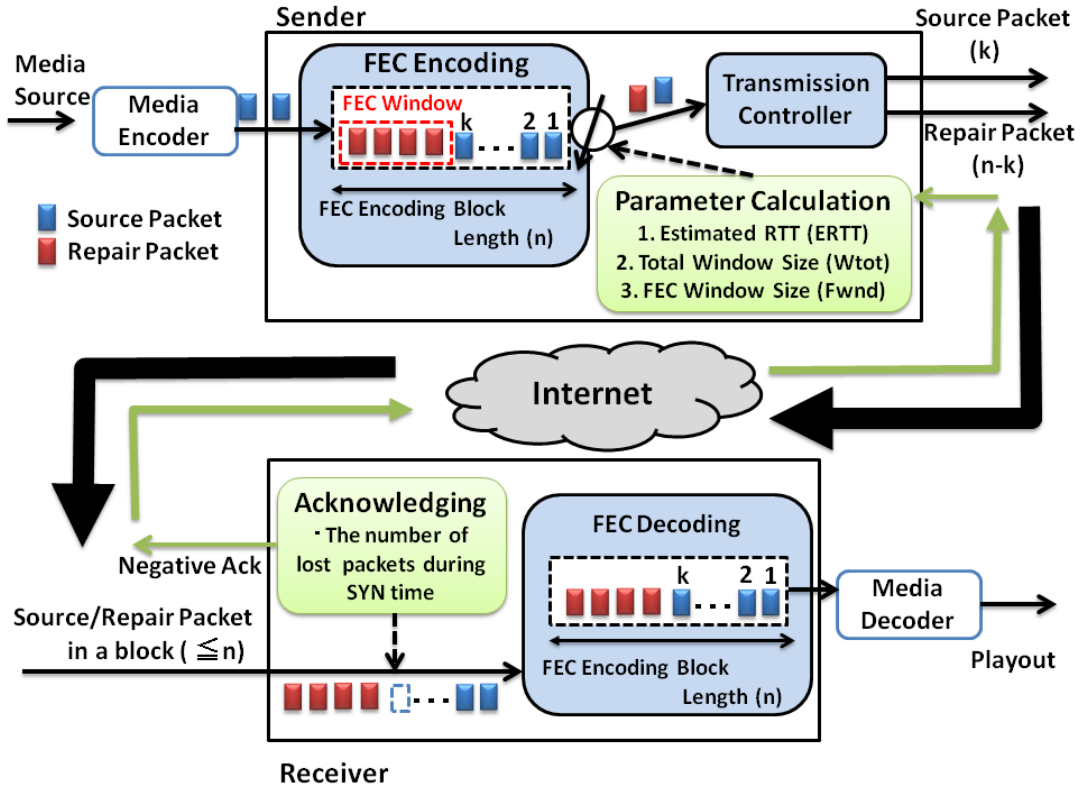


Figure 5.1: **GENEVA overview.**

Fig. 5.1 illustrates the GENEVA overview. GENEVA operates mainly at the sender end. A sender transmits data and FEC repair packets over a real-time transport protocol (RTP) [6] carried on top of the Internet Protocol (IP) and UDP. An GENEVA sender collects the feedback information transmitted over a real-time transport control protocol (RTCP) delivered by a receiver, then adjusts the degree of FEC redundancy based on packet loss conditions and estimated RTT. The data transmission rate, which depends on the video format preliminarily assigned, is maintained during transmission.

GENEVA leverages an AL-FEC, in which $n - k$ FEC repair packets are added to a block of k data packets (FEC source packets). We consider maximum distance separable (MDS) codes such as Reed-Solomon codes [7] which can recover all of the missing packets from any set of exactly k packets. Here, we define the number of FEC repair packets as the "FEC window size $Fwnd$ (packets)", indicated by

"$n - k$". If the code parameters, the FEC window size and n, are appropriately set in the event of packet losses, a receiver may recover all of the missing data packets within a block. To calculate and create FEC repair packets in each block, all of the generated data packets (FEC source packets) are stored once in the FEC encoding block buffer. Depending on the FEC window size, the number of FEC repair packets stored in the FEC encoding block buffer varies.

5.2 Algorithm and Analysis

We now describe the GENEVA algorithm and its parameter settings based on the aforementioned design criteria.

5.2.1 Acknowledging

To control network congestion by adjusting the $Fwnd$, an GENEVA sender needs negative acknowledgements of packet reception from a corresponding receiver. Since GENEVA tolerates moderate network congestion without reductions in the data transmission rate (unlike TCP), generating negative acknowledgements may consume large bandwidth. To reduce the ratio of bandwidth consumed by control traffic, an GENEVA receiver sends feedback information at constant time interval (SYN), which denotes the number of lost packets during SYN time. The SYN is related to responsiveness or stability of GENEVA flow (i.e., conditions of FEC recovery capabilities). Note that since GENEVA stores the k source packets in each block to generate the $Fwnd$ repair packets, it takes a certain time for a sender to adjust and send $Fwnd$ repair packets after the last adjustment. GENEVA thus maintains the FEC window size greater than or equal to the minimum FEC window size ($Fwnd_{min}$) to allow the flow to recover lost source packets in moderate network congestion. In a situation in which feedback information is lost, GENEVA performance (i.e., FEC recovery capabilities) is not sensitively affected due to the preliminarily operated $Fwnd$. In addition, GENEVA maintains the current $Fwnd$ without increasing the $Fwnd$, and therefore does not induce an adverse impact on performance of other competing flows. In GENEVA, SYN time is currently set to 0.01 sec based on our comprehensive simulation results.

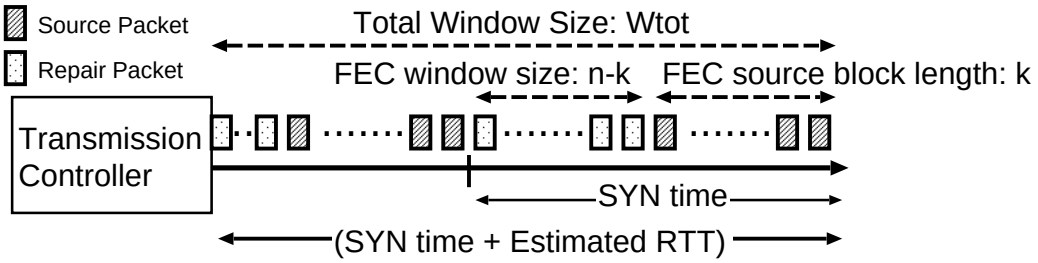


Figure 5.2: **FEC source and repair transmission.**

5.2.2 FEC Source and Repair Transmission

Fig. 5.2 shows the transmission of FEC source and repair packets. GENEVA adjusts the $Fwnd$ in an arbitrarily fixed FEC source block length (k) during transmission. The value of k is chosen to be the number of FEC source packets needed for a sender to transmit during SYN time (0.01 sec), which depends on the video format and its encoding parameters. For instance, on the assumption that a sender transmits RTP/HDV (MPEG2-TS) packets of 1500 bytes at a rate of 25 Mbps [14], the value of k is calculated as follows: $k = 25(Mbps) \times 10^6 \times 0.01(sec)/(1500 \times 8) = 20.83$. Thus, in the case of transmission RTP/HDV, the value of k becomes 20 or 21. The generated FEC repair packets for each source block are sent together with the corresponding source packets within SYN time. Thus, each time a sender receives feedback information, $Fwnd$ is adjusted using the increase/decrease algorithm (described below). It is well understood from queuing theory that burst transmission of packets poses bursty packet loss that limits FEC recovery performance [79, 12]. To avoid a generation of bursty traffic, the transmission controller evenly spaces FEC source and repair packets sent into the network over SYN time by controlling the inter-packet gap (IPG). The IPG time during an arbitrary SYN time becomes as follows: $IPG = \mathrm{SYN}/(Fwnd + k)$.

Using feedback information, a sender calculates the sample RTT ($SRTT$) to derive the number of unacknowledged packets sent to the network, which is denoted by the "current total window size (W_{tot})". The W_{tot} is used to adjust the $Fwnd$ with the increase/decrease algorithm. Keeping pairs of the sequence number and its transmitted time, a sender calculates $SRTT$. Using an exponential weighted moving average with the smoothing factor of α ($0 < \alpha < 1$), the estimated RTT

($ERTT$) is calculated as follows: $ERTT = \alpha \times ERTT + (1 - \alpha) \times SRTT$. Since the W_{tot} denotes the number of unacknowledged packets sent to the network (i.e., the number of packets sent to the network during $ERTT$ plus SYN time), the W_{tot} is dynamically derived using $ERTT$, SYN time and the current IPG, as follows:

$$\begin{aligned} W_{tot} &= (ERTT + \text{SYN})/IPG \\ &= (Fwnd + k) \times (ERTT + \text{SYN})/\text{SYN} \end{aligned} \tag{5.1}$$

5.2.3 FEC Window Size Adjustment

We define a controller that forces the total window size to achieve an equilibrium point (W^*_{tot}) where the $Fwnd$ stably converges at a packet loss probability $p < p_{max}$:

$$W^*_{tot} = a/(p_{max} - p) \tag{5.2}$$

where a is a constant with $a > 0$, and is the scaling property of W_{tot}. This equation shows that, as the network becomes more congested (i.e., p increases), W^*_{tot} also becomes larger, which results in an increase in the $Fwnd$. Since we consider a high-performance real-time application that consumes at least 25 Mbps (W_{tot} becomes about 42 under 10 ms RTT), we set a to 2 to scale up the $Fwnd$. Using Eq.(5.2), the $Fwnd$ in the steady state is shown as follows:

$$Fwnd = \lfloor (W^*_{tot} \times \text{SYN}/(ERTT + \text{SYN}) - k \rfloor \tag{5.3}$$

When the $Fwnd < Fwnd_{min}$, GENEVA maintains the $Fwnd$ at $Fwnd_{min}$. Note that although a larger W_{tot} that a flow has from the first (e.g., with high throughput and/or longer RTT) cannot approach the W^*_{tot}, the $Fwnd$ converges at small sizes, which contributes to stability of other competing GENEVA flows.

GENEVA adopts the Generalized Multiplicative Increase and Additive Decrease (GMIAD) algorithm to achieve an equilibrium point in Eq.(5.3). When a sender receives feedback information indicating the number of both lost and delivered packets during SYN time, the following calculation is repeated for a number

of times equal to the number of lost packets, to first update the W_{tot}:

$$W_{tot} \leftarrow W_{tot} + i(W_{tot}) \tag{5.4}$$

where $i(W_{tot})$ is an incremental function of W_{tot}. Then, the W_{tot} is derived by repeating the following calculation for a number of times equal to successfully delivered packets:

$$W_{tot} \leftarrow W_{tot} - b \tag{5.5}$$

where b is a constant with $b > 0$.

The behavior of the aforementioned GMIAD algorithm is described with a differential equation:

$$\frac{dW_{tot}}{dt} = i(W_{tot})\frac{W_{tot}}{(\mathrm{RTT} + \mathrm{SYN})}p - b\frac{W_{tot}}{(\mathrm{RTT} + \mathrm{SYN})} \tag{5.6}$$

At an equilibrium point, the following equation is derived by Eq.(5.6):

$$i(W^*_{tot}) = \frac{b}{p} \tag{5.7}$$

Using Eqs.(5.2) and (5.7), $i(W_{tot})$ is given with the constant value p_{max} and b:

$$i(W_{tot}) = bW_{tot}/(p_{max}W_{tot} - 2) \tag{5.8}$$

5.2.4 Parameter Settings

The value of α for the estimated RTT ($ERTT$) calculation determines how rapidly $ERTT$ adapts to the sample RTT ($SRTT$) changes. $ERTT$ using a lower value of α adapts to $SRTT$ changes more rapidly. In a situation in which $SRTT$ fluctuates significantly, $ERTT$ using a lower value of α also fluctuates significantly. This situation produces large fluctuations in W_{tot}, and the fluctuated W_{tot} determines the $Fwnd$ irrespective of packet loss conditions. Since as shown in Eq.(5.2), the equilibrium point of W_{tot} is defined by using packet loss probability (p), such a

situation may cause improper adjustments of the $Fwnd$. To avoid it, α is set to relatively high value, 0.9.

The setting of p_{max} defines the upper limit of packet loss probability that an GENEVA flow can tolerate. Because GENEVA with a high value of p_{max} increases the Fwnd up to a maximum extent under high loss rate regimes (i.e., packet loss probability is around the p_{max}), the value of p_{max} relates to friendliness for competing TCP flows and GENEVA aggressiveness. Since packet loss plays a key role in achievable TCP performance [64, 85], GENEVA with high value of p_{max} thus may cause an adverse impact on competing TCP performance or congestion collapse. Conventional wisdom holds that a loss rate of more than 5% has a significant adverse effect on TCP performance, because it will greatly limit the size of the congestion window and hence the transfer rate, while 3% is often substantially less serious [3]. Thus, p_{max} of upper limit of packet loss probability that GENEVA flow can tolerate is set to 0.03 in this work.

When $Fwnd_{min}$ is set to $\lceil k \times p_{max} \rceil$ in association with p_{max}, GENEVA with the k of about 20 keeps the $Fwnd$ of 1 in the absence of packet loss and cannot recover lost data packets by sudden bursty packet loss. Therefore, $Fwnd_{min}$ is set to a constant value of 8 in this work. Also, it is important to decide the maximum acceptable $Fwnd$, because the additional delay by FEC encoding/decoding becomes large with the increase in 1) k and the $Fwnd$ (i.e., encoding time) and 2) packet loss rate (i.e., the decoding time becomes large with the increase in the number of recovered packets within a block). According to the work using a real implementation with Reed-Solomon codes for a high-performance real-time application transmitting at a rate of 30 Mbps [14], the additional video frame delay compared to the case without FEC becomes lower about 150 ms within less than 15% of packet loss rate, where the value of k and the number of repair packets within a block were set to 170 and 85 respectively. On the assumption that the value of k that GENEVA holds becomes smaller than 170, the maximum acceptable $Fwnd$ of GENEVA is set to 60 so as to suppress the additional delay to below 150 ms. GENEVA thus requires the equivalent buffer length of about 150 ms at the cost of the real-time performance. If the packet loss probability continues to be more than or equal to p_{max}, GENEVA keeps the maximum acceptable $Fwnd$ in the presence of the high packet loss rate, which results in the significant degradation of competing TCP performance and congestion collapse. GENEVA thus

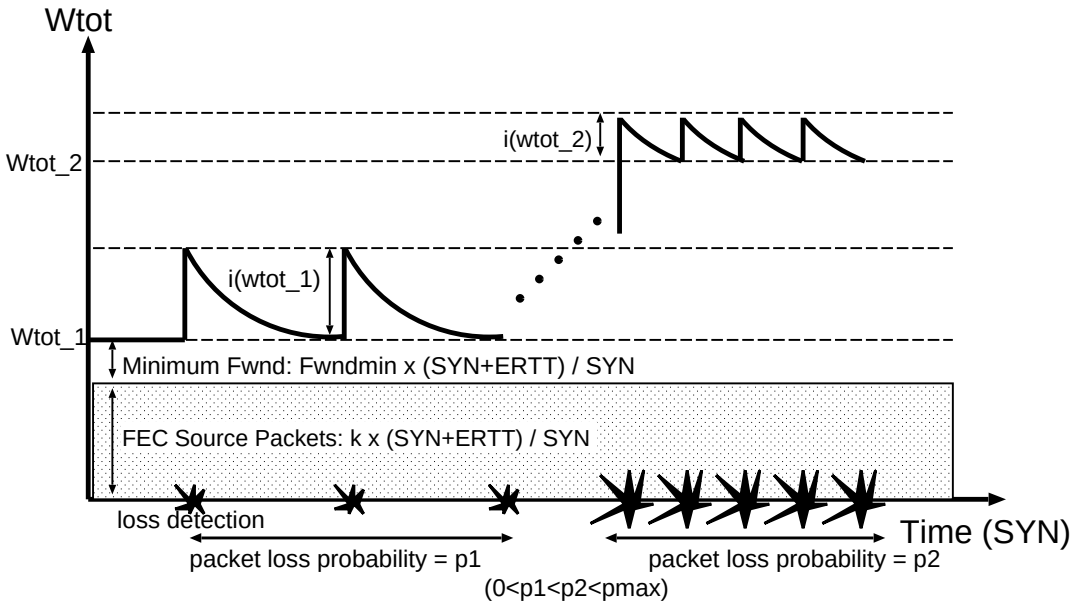


Figure 5.3: **GENEVA scaling properties.**

needs to reduce the data transmission rate at the cost of video quality to avoid congestion collapse. In this study, we do not assume such a situation where the physical bandwidth is notably less than the averaged consumption bandwidth of TCP and high-performance streaming flows.

Fig. 5.3 shows the GENEVA scaling properties of W_{tot} when the packet loss probability $p = p1, p2$ $(0 < p1 < p2 < p_{max})$. Whereas W_{tot} increases by $i(W_{tot})$ in response to packet loss, W_{tot} decreases based on Eq.(5.5) during periods of no packet loss. Although the W_{tot} in $p2$ (W_{tot}_2) becomes larger than the W_{tot} in $p1$ (W_{tot}_1), both the increase ratio $i(W_{tot}\_2)$ and decrease speed up to W_{tot}_2 become smaller compared those in $p1$.

The increase in $Fwnd$ based on $i(W_{tot})$ should be adjusted properly to minimize an adverse effect on competing TCP performance by utilizing network resources that TCP fails to copy. In TCP, the window size (W_{tcp}) is halved on detecting a packet loss during a round trip time, and becomes $W_{tcp}/2$. The constant value b, which relates to $i(W_{tot})$ as shown in Eq.(5.8), should be set to prevent $i(W_{tot})$ from exceeding the decrease in TCP window size (namely, $W_{tcp}/2$), in order to achieve stable conditions. Fig. 5.4 shows both $i(W_{tot})$ and $W_{tcp}/2$ as a function

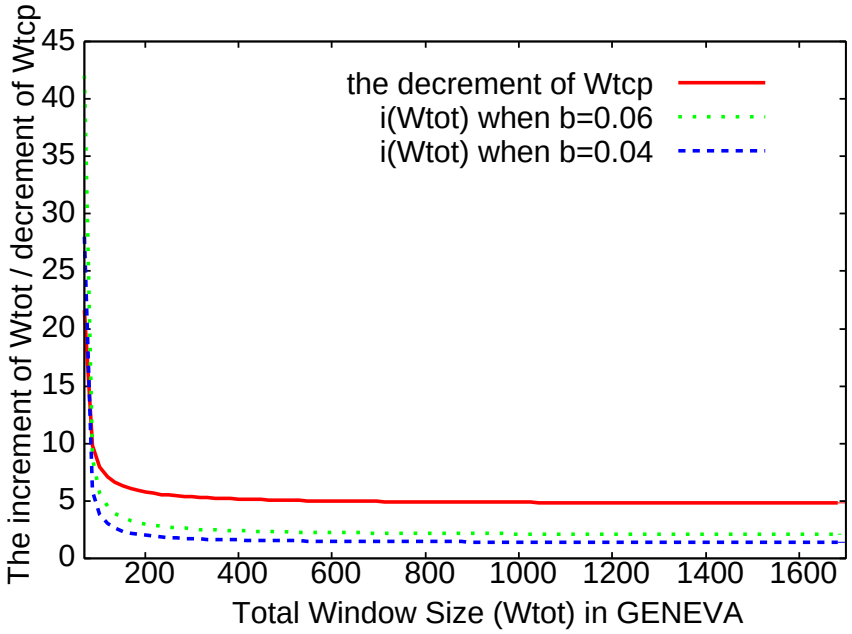


Figure 5.4: **The increment of W_{tot} ($i(W_{tot})$) and decrement of TCP window size (W_{tcp}) as a function of W_{tot}. From the aspect of design that $i(W_{tot})$ should not exceeds the corresponding decrement of $Wtcp$, the constant b (decrement function in GMIAD algorithm) is set to 0.04.**

of W_{tot}, by using the TCP performance model [64]. The range of W_{tot} is set to between 70 and 1700. The W_{tot} value of 1700 represents about the size of W_{tot} that GENEVA at a rate of 25 Mbps achieves with the maximum acceptable $Fwnd$ (i.e., $Fwnd = 60$) under RTT 200 ms: $W_{tot} = (60+21) \times (0.2+0.01)/0.01 = 1701$, as shown in Eq.(5.1). When W_{tot} of larger value than $2/p_{max}$ approaches $2/p_{max}$ (i.e., about 66.6), $i(W_{tot})$ approaches positive infinity (as shown in Eq.(5.8)). The range of W_{tot} thus starts from 70. When b becomes large, GENEVA becomes aggressive to increase the $Fwnd$ with an increase in the W_{tot}. As you can see in Fig. 5.4, each $i(W_{tot})$ with b=0.06 and 0.04 does not exceed $W_{tcp}/2$ at an almost full range of W_{tot}. However, each $i(W_{tot})$ with b=0.06 and 0.04 exceeds the $W_{tcp}/2$ in below about 80 and 73 of W_{tot}, respectively. In this work, we set b to 0.04. In below 73 of W_{tot}, $i(W_{tot})$ is constantly set to 10 so as not to exceed $W_{tcp}/2$. Note that although $i(W_{tot})$ in large W_{tot} becomes small, GENEVA tends to gradually increase the $Fwnd$ in conditions of bursty packet loss by reacting to every packet loss event.

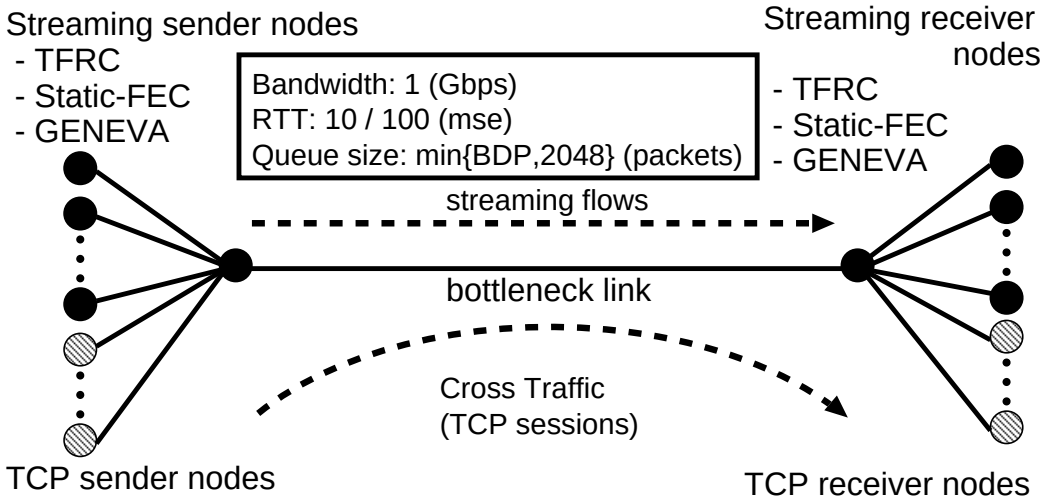


Figure 5.5: **The simulation topology (single bottleneck link).**

5.3 Evaluation

5.3.1 Simulation Setup and Performance Metrics

Using an NS-2 extended with GENEVA module, we performed experiments using the dumbbell topology (i.e., single bottleneck link) and the multi-branch topology (i.e., multiple bottleneck links), as shown in Fig. 5.5 and 5.12. The bandwidth of the bottleneck link was set to 1 Gbps. Each sender and receiver were connected to the bottleneck link through the 10 Gbps access link with propagation delay of about 1 ms. The two-way propagation delay (i.e., minimum RTT, RTTmin) was set to 10 or 100 ms. According to RTTmin settings, the propagation delay in the bottleneck link varies, and BDP is calculated using the RTTmin. The packet size was set to 1500 bytes for all connections. A drop-tail queuing was used at the router in the bottleneck link, the queue length size was set to min{BDP, 2048 (packets)}. The queue length size in RTTmin of 10 ms was set to BDP of setting RTTmin 10 ms, where RTT becomes the RTTmin and twice the RTTmin. In RTTmin of 100 ms, BDP in packets is calculate as follows: $BDP = 1(Gbps) \times 10^9 \times 0.1(sec)/(1500 \times 8) = 8333.33$. Since the value of the queue size in packets exceeds 2048, the queue size in RTTmin of 100 ms was set to 2048, where RTT becomes between the RTTmin and the RTTmin plus maximum queuing delay (about 25

ms). About 25% of the BDP is available as buffers on the bottleneck link when the RTTmin is 100 ms; A decrease in available queuing delay is favorable in terms of the cost of implementing high-speed memory systems in network devices, and also for end system applications. Each simulation ran for about 3 minutes.

As the GENEVA performance metrics, we observed 1) the average residual data loss rate of GENEVA flows which denotes the ratio of the total number of non-recovered data packets to the total number of sent data packets, 2) the average number of "bursty packet loss events", and 3) the average throughput of competing TCP flows. We here define the bursty packet loss event as the situation in which more than three packets are consecutively lost in an GENEVA flow, and expect that GENEVA suppresses the number of occurrence of bursty packet loss event to improve FEC recovery capabilities.

The metrics of the average residual data loss rate and the average number of bursty packet loss events were compared with those observed when we used small/large Static-FEC flows (i.e., the $Fwnd$ is constant) and DP-FEC flows [60] under the same network condition. We set the $Fwnd$ of small and large Static-FEC flow to $Fwnd_{min}$ and 1.5 times the $Fwnd_{min}$, respectively. The source block length of DP-FEC is set to the same value as that of GENEVA. As TCP Performance index ($TPindex$), we use the result of the average throughput of TCP flows observed when they compete with TFRC flows, and define $TPindex$ as follows:

$$TPindex = \frac{TCPthuput\_Target}{TCPthuput\_TCPstr} \tag{5.9}$$

where $TCPthuput\_Target$ is the result of the average throughput of TCP flows competing with small/large Static-FEC or DP-FEC or GENEVA flows; $TCPthuput\_TCPstr$ is the result of the average throughput of TCP flows competing with TFRC flows under the same network condition.

We used two types of TCP flows using TCP-SACK, 1) short-lived TCP flows and 2) long-lived TCP flows. Short-lived TCP flows arrive at the bottleneck link, as a Poisson process with an average rate of r_tcp flows per second. The size of each TCP flow follows Pareto distribution with an average of s_tcp packets and shape parameter 1.5. We define the load of TCP flows as $\rho\_tcp = r\_tcp \times s\_tcp$. Long-lived TCP flows are persistent in the network.

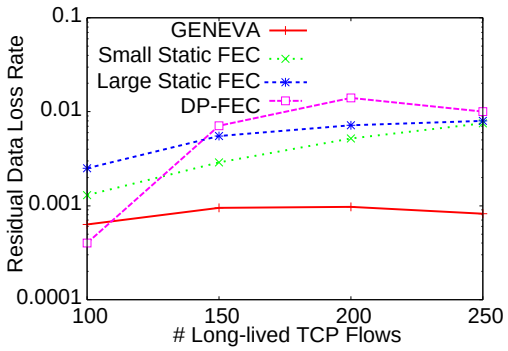
5.3.2 Homogeneous GENEVA Flows VS. TCP Flows in Single Bottleneck Link

In this experiment, all GENEVA or Static-FEC flows transmit data packets at a rate of 30 Mbps in the single bottleneck link. We set the number of streaming flows to 10, and evaluated the GENEVA performance in competition with long-lived TCP flows or short-lived TCP flows in network conditions where the total packet loss rate on the bottleneck link varies from about 1% to 4%. This range of the packet loss ratio is based on the 2012 yearly reports of the ICFA-SCIC Monitoring WG [3] indicating that the packet loss rate on the Internet is around 1% on average.

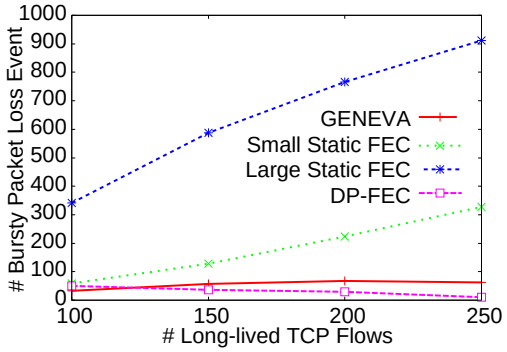
Competition with Long-lived TCP Flows

Figs. 5.6 and 5.7 shows the results of the experiments where 10 GENEVA flows or 10 small/large Static-FEC flows or 10 DP-FEC flows compete with long-lived TCP flows under the RTTmin 10 ms and 100 ms, respectively. Under both of the RTTmin (10 and 100 ms), as the number of competing TCP flows increases, the number of bursty packet loss events of the GENEVA flows becomes lower than that of the small Static-FEC flows. Therefore, the corresponding residual data loss rate also decreases. The $TPindex$ of both small Static-FEC and GENEVA flows becomes about 0.65, because they preserve the highest data transmission rate (i.e., 30 Mbps) unlike TFRC. Although the GENEVA flows increase their $Fwnd$ in response to packet losses, the $TPindex$ is almost the same as that of the small Static-FEC flows (about 0.65). This means that the GENEVA flows effectively utilized network resources that TCP flows fail to copy, by properly adjusting the $Fwnd$.

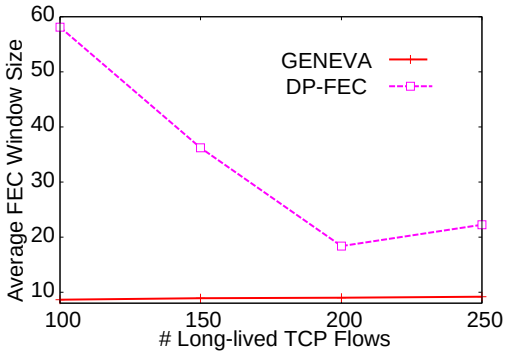
As shown in Fig. 5.6(b), the large Static-FEC flows under the RTTmin 10 ms cause more bursty packet loss events than that of small Static-FEC flows or GENEVA flows, which results in higher residual data loss rate of the large Static-FEC flows. In addition, as seen from Fig. 5.6(d), the corresponding $TPindex$ of the large static-FEC flows becomes lower than that of GENEVA flows due to the constantly added large FEC redundancy. In the case of the RTTmin 100 ms, since TCP flows become less aggressive compared to the case of the RTTmin 10



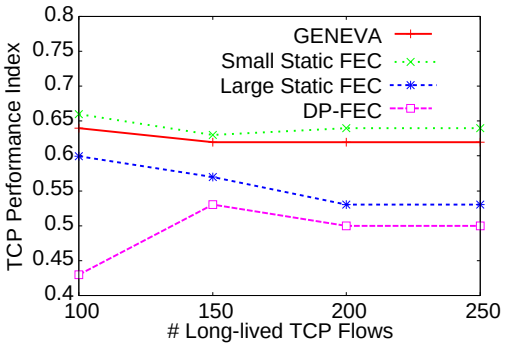
(a) Residual data loss rate (RTTmin 10 ms)



(b) The number of bursty packet loss event (RTTmin 10 ms)



(c) Average FEC window size (RTTmin 10 ms)

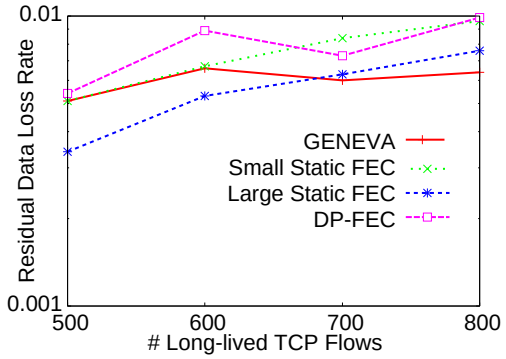


(d) TCP performance index (RTTmin 10 ms)

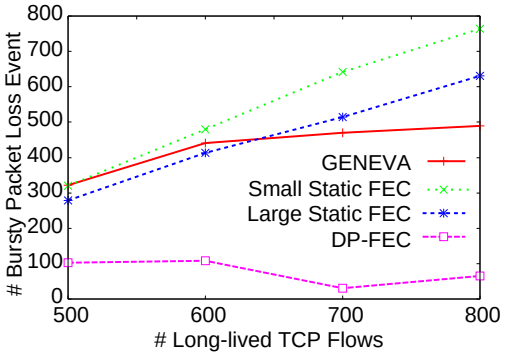
Figure 5.6: **The results of the performance of 10 GENEVA flows under RTTmin 10 ms in competition with long-lived TCP flows. Three metrics (residual data loss rate, the number of bursty packet loss events and TCP performance index) were compared to those of 10 small/large Static-FEC flows and 10 DP-FEC flows under the same network condition.**

ms, the number of bursty packet loss events of the large Static-FEC flows becomes lower than that of the small Static-FEC flows, and the corresponding residual data loss rate also becomes lower (as seen from Figs. 5.7(b) and 5.7(a)). Although the residual data loss rate of the large Static-FEC flows competing with 500 or 600 TCP flows becomes lower than that of the GENEVA flows, the $TPindex$ of the large Static-FEC flows becomes lower than that of GENEVA flows (as seen from Fig. 5.7(d)).

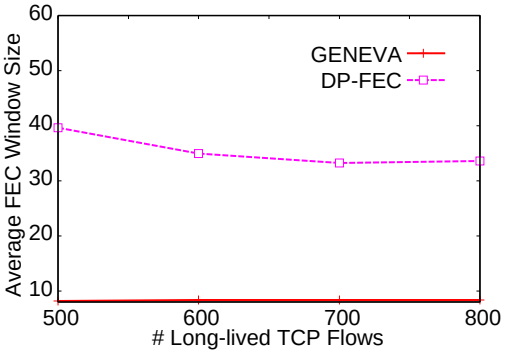
Since DP-FEC probes the available bandwidth by increasing the $Fwnd$, the DP-



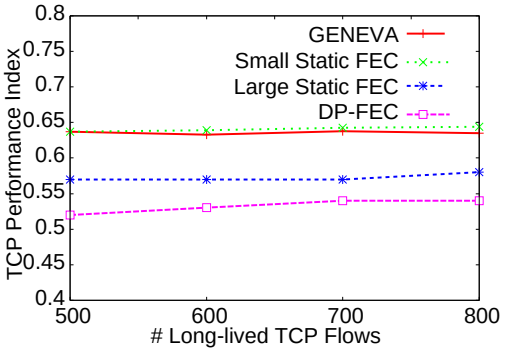
(a) Residual data loss rate (RTTmin 100 ms)



(b) The number of bursty packet loss event (RTTmin 100 ms)



(c) Average FEC window size (RTTmin 100 ms)



(d) TCP performance index (RTTmin 100 ms)

Figure 5.7: **The results of the performance of 10 GENEVA flows under RTTmin 100 ms in competition with long-lived TCP flows. Three metrics (residual data loss rate, the number of bursty packet loss events and TCP performance index) were compared to those of 10 small/large Static-FEC flows and 10 DP-FEC flows under the same network condition.**

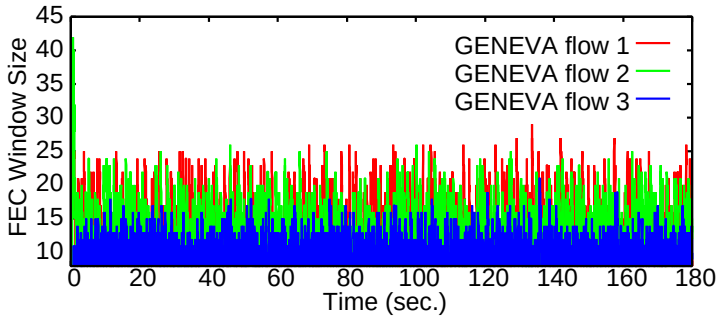
FEC flows under the RTTmin 10 ms and 100 ms maintain a much larger $Fwnd$, compared to that of the GENEVA flows (as seen from Fig. 5.6(c) and 5.7(c)). However, since their increased $Fwnd$ induces residual data packet loss as a side effect between the competing DP-FEC flows, the residual data loss rate becomes higher than that of the GENEVA flows, except in the case where the DP-FEC flows under the RTTmin 10 ms compete with 100 TCP flows (as seen from Fig. 5.6(a) and 5.7(a)). Moreover, due to the large average $Fwnd$, the $TPindex$ of the DP-FEC flows becomes lower than that of large Static-FEC flows.

Whereas the average $Fwnd$ of the GENEVA flows under the RTTmin 10 ms increases up to about 9.2 (in competition with 250 TCP flows), the $Fwnd$ under the RTTmin 100 ms maintains almost the same value (between 8.2 and 8.4). This is because GENEVA flows under the longer RTT have larger total window size. This behavior contributes to the stability. Fig. 5.8 shows the trace of the $Fwnd$ of the selected 3 GENEVA flows, and the packet loss rate in the bottleneck link averaged over RTTmin interval and also its total packet loss rate. Because each of packet loss rates that the GENEVA flows observed is different, the variation of the corresponding $Fwnd$ is also different. However, under both of the RTTmin, the GENEVA flows keep both of the $Fwnd$ of each flow and the link loss rate stable.

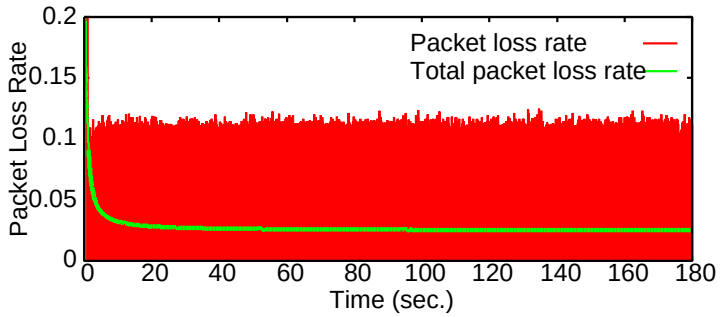
Competition with Short-lived TCP Flows

To investigate the effect of TCP slow-start on the GENEVA performance, we set the experiments where 10 streaming flows compete with short-lived TCP flows. The load of short-lived TCP flows (ρ_tcp) was set to 50% of the bottleneck link capacity, and the rate of an occurrence of TCP flow per second (r_tcp) was set to 12.5 or 25. Table 5.1 shows the results of the experiments. Under the RTTmin 10 ms, the GENEVA flows suppress the number of bursty packet loss events, compared to that of small/large Static-FEC flows. The suppression contributes to the reduction in the residual data loss rate. The $TPindex$ of the GENEVA flows becomes slightly lower than that of the small Static-FEC flows and becomes higher than that of the large Static-FEC flows, because the added FEC redundancy effectively prevents the congestion window size of TCP flows from increasing aggressively. Since the DP-FEC flows under the RTTmin 10 ms has the large average $Fwnd$, the $TPindex$ decreases by more than 0.4 compared to that of the GENEVA flows in both of r_tcp =12.5 and 25. In addition, the residual data loss rate of the DP-FEC flows becomes higher than that of the GENEVA flows.

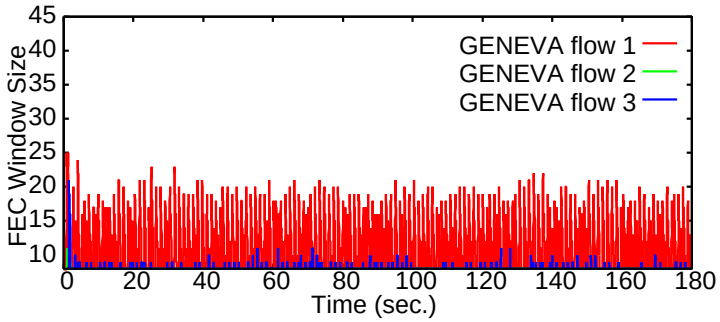
On the other hand, under the RTTmin 100 ms, the data loss rate of the GENEVA flows was not significantly improved especially in r_tcp =25, due to the large total window size caused by the longer RTTmin 100 ms. However, the GENEVA flows reduce the number of bursty packet loss events, and maintain almost the same $TPindex$ as the small Static-FEC flows and higher $TPindex$ than that of the large Static-FEC flows. Although the residual data loss rate of the



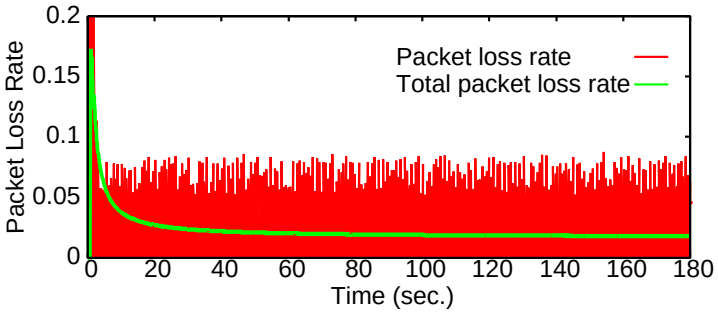
(a) FEC window size (RTTmin 10 ms)



(b) Packet loss rate on the bottleneck link (RTTmin 10 ms).



(c) FEC window size (RTTmin 100 ms)



(d) Packet loss rate on the bottleneck link (RTTmin 100 ms)

Figure 5.8: **Trace of FEC window size of three selected GENEVA flows and packet loss rate on the bottleneck link. 10 GENEVA flows under RTTmin 10 and 100 ms compete with 200 and 700 long-lived TCP flows, respectively.**

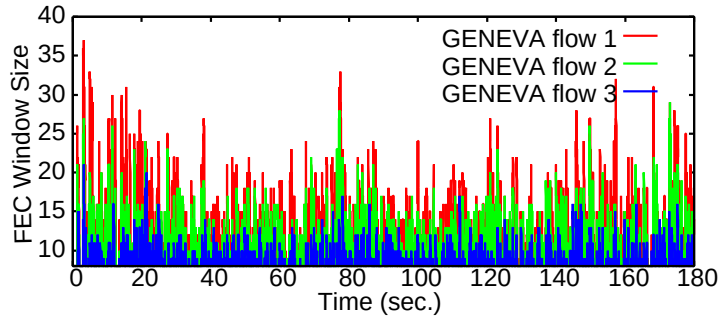
Table 5.1: **The results of 10 GENEVA flows, 10 small/large Static-FEC flows and 10 DP-FEC flows under RTTmin 10 and 100 ms in competition with short-lived TCP flows.**

TCP load(ρ_tcp): 500Mbps	Data Rate (Mbps)	Data Loss Rate	Packet Loss Rate	FEC Window Size	Total Window Size	# Bursty Packet Loss Event	TCP performance index	Link Loss Rate
RTTmin:10 ms, $r\_tcp = 12.5$	avg.	avg.	avg.	avg.	avg.	avg.	avg.	avg.
10 GENEVA flows	30.0	0.0011	0.0068	10.19	105.26	164.90	0.57	0.039
10 Small Static-FEC flows	30.0	0.0029	0.0078	8.0	—	305.70	0.61	0.040
10 Large Static-FEC flows	30.0	0.0026	0.0083	12.0	—	281	0.49	0.039
10 DP-FEC flows	30.0	0.0024	0.32	48.12	—	169.7	0.10	0.024
RTTmin:10 ms, $r\_tcp = 25$								
10 GENEVA flows	30.0	0.0013	0.0070	9.88	110.56	283.60	0.61	0.033
10 Small Static-FEC flows	30.0	0.0029	0.0089	8.0	—	509.40	0.65	0.035
10 Large Static-FEC flows	30.0	0.0032	0.0093	12.0	—	938.5	0.46	0.037
10 DP-FEC flows	30.0	0.0061	0.23	35.69	—	610	0.18	0.025
RTTmin:100 ms, $r\_tcp = 12.5$								
10 GENEVA flows	30.0	0.0039	0.0074	8.19	424.34	393.50	0.77	0.030
10 Small Static-FEC flows	30.0	0.0044	0.0077	8.0	—	452.0	0.78	0.030
10 Large Static-FEC flows	30.0	0.0050	0.0093	12.0	—	549.5	0.63	0.031
10 DP-FEC flows	30.0	0.0016	0.35	58.11	—	878.6	0.22	0.025
RTTmin:100 ms, $r\_tcp = 25$								
10 GENEVA flows	30.0	0.0036	0.0069	8.14	425.41	459.60	0.82	0.023
10 Small Static-FEC flows	30.0	0.0037	0.0070	8.0	—	473.20	0.82	0.024
10 Large Static-FEC flows	30.0	0.0036	0.0079	12.0	—	688.6	0.70	0.027
10 DP-FEC flows	30.0	0.0021	0.25	40.41	—	1186.0	0.33	0.030

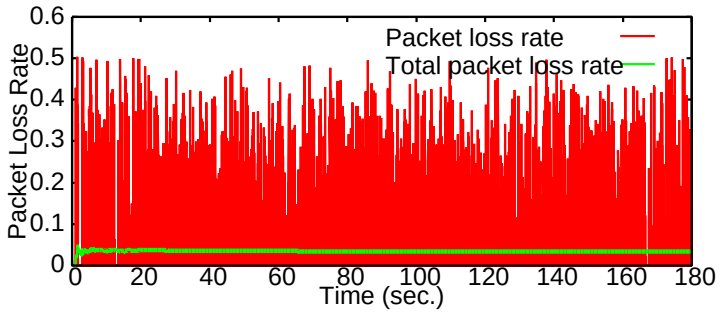
DP-FEC flows in both of r_tcp =12.5 and 25 under the RTTmin 100 ms becomes lower than that of the GENEVA flows, the $TPindex$ of the DP-FEC flows decreases considerably by more than 0.49. Thus, the DP-FEC flows fail to prevent adverse effects on TCP performance. However, the residual data loss rate of the GENEVA flows becomes less than or equal to small/large Static-FEC flows, and the GENEVA flows maintain almost the same $TPindex$ as the small Static-FEC flows by having the function of the total window size control to adjust the $Fwnd$.

Fig. 5.9 shows the trace of one of the results in the experiments. We can see that, although the packet loss rate fluctuates and bursty packet losses occur, both of the $Fwnd$ of the GENEVA flows and the total packet loss rate on the bottleneck link become stable.

Note that since 1) the number of bursty packet loss events and residual data loss rate of the GENEVA flows became less than that of the small Static-FEC flows and 2) the $TPindex$ of GENEVA flows became almost the same as that of the small Static-FEC flows, the increased $Fwnd$ did not induce more residual data packet loss and degradation of $TPindex$ as a side effect for the competing flows. However, when there is not fully available bandwidth in a congested link (e.g., in



(a) FEC window size

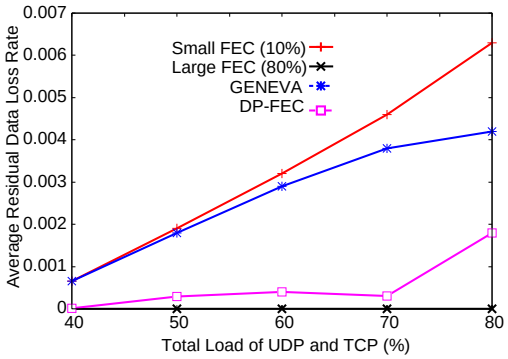


(b) Packet loss rate on the bottleneck link

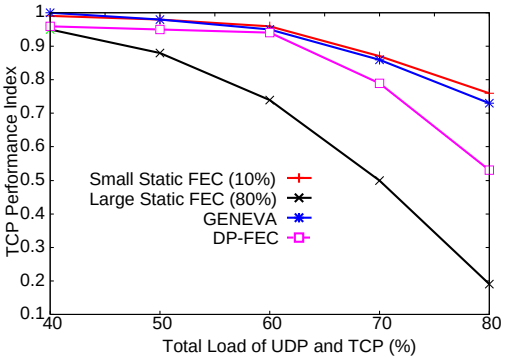
Figure 5.9: **Trace of FEC window size of three selected GENEVA flows competing with short-lived TCP flows under RTTmin 10 ms, and packet loss rate on the bottleneck link. In this experiment, the load of TCP flows (ρ_tcp) was set to 50% of the bottleneck link capacity, and the rate of an occurrence of TCP flow per second (r_tcp) was set to 25.**

case that the bottleneck link bandwidth is much less than 1 Gbps), the increased $Fwnd$ induces packet loss up to more than p_{max}, where the residual data loss rate increases and the $TPindex$ degrades. In such condition, it would be necessary to reduce the data transmission rate and degrade the video quality, which we have not assumed as described in Section 1.2.

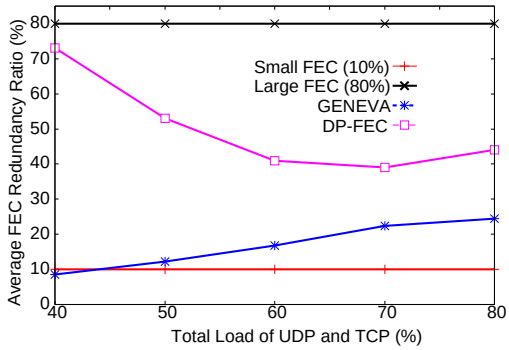
To simplify a comparison with the DP-FEC performance according to TCP load, we performed experiments under the same network condition as in Section 4.3.4. The all parameters of DP-FEC and Static-FEC remain. Fig. 5.10 shows the results of the experiments. Because the GENEVA flows add higher degree of FEC redundancy than that of the small Static-FEC flows under the total UDP and load of more than 50% as shown in Fig. 5.10(c), the average residual data loss rate of the GENEVA flows decrease more than that of the small Static-FEC flows



(a) Average residual data loss rate



(b) TCP performance index



(c) Average FEC redundancy ratio (%)

Figure 5.10: **Comparison with DP-FEC and Static-FEC performances when the total load of UDP flows (10 streaming flows without FEC) and TCP flows is between 40% and 70%.**

as shown in Fig. 5.10(a). Nevertheless, due to their FEC window size adjustments, the GENEVA flows achieve $TPindex$ as high as that the small Static-FEC flows irrespective of TCP load, as shown in Fig. 5.10(b). Since DP-FEC aggressively increases the $Fwnd$ without supporting window control, the average residual data loss rate of the DP-FEC flows is suppressed significantly. However, $TPindex$ of the DP-FEC flows decreases especially under high total loads of UDP and TCP. In addition, although the large Static-FEC flows considerably decrease the average residual data loss rate, $TPindex$ ot their flows highly decreases due to the large consumption bandwidth of FEC. Although DP-FEC suppresses the residual data loss rate and achieves relatively high $TPindex$ under low TCP load, DP-FEC cannot achieve higher $TPindex$ than GENEVA under high TCP load. Therefore, since GENEVA achieve higher $TPindex$ than DP-FEC irrespective of TCP load

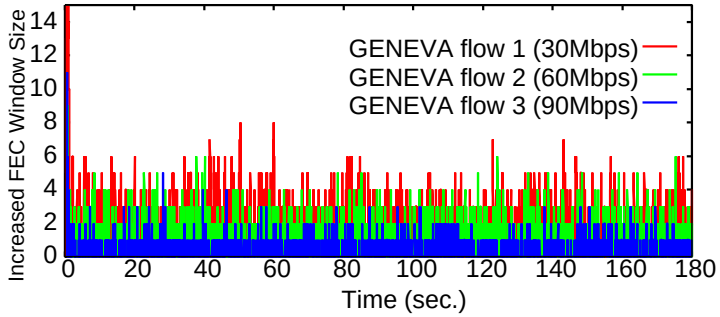
Table 5.2: **The results of 7 GENEVA flows, 7 small/large Static-FEC flows and 7 DP-FEC flows with different data transmission rates under RTTmin 10 ms, in competition with 200 long-lived TCP flows.**

	Data Rate (Mbps)	Data Loss Rate	Packet Loss Rate	FEC Window Size	Total Window Size	# Bursty Packet Loss Event	TCP performance index	Link Loss Rate
RTTmin:10 ms # TCP flows:200	avg.	avg.	avg.	avg.	avg.	avg.	avg.	avg.
5 GENEVA flows	30.0	0.0014	0.012	9.28	143.75	110.6	—	—
1 GENEVA flow	60.0	0	0.0095	9.87	273.35	29	—	—
1 GENEVA flow	90.0	0	0.011	9.21	413.25	4	—	—
Total (Avg.)	42.9	0.0010	0.011	9.35	200.77	83.71	0.62	0.026
5 Small Static-FEC flows	30.0	0.0027	0.020	8.0	—	416.4	—	—
1 Small Static-FEC flow	60.0	0	0.0089	8.0	—	13	—	—
1 Small Static-FEC flow	90.0	0	0.011	8.0	—	0	—	—
Total (Avg.)	42.9	0.0039	0.016	8.0	—	299.29	0.63	0.027
5 Large Static-FEC flows	30.0	0.0072	0.023	12.0	—	873	—	—
1 Large Static-FEC flow	60.0	0	0.0080	12.0	—	2	—	—
1 Large Static-FEC flow	90.0	0	0.0096	12.0	—	2	—	—
Total (Avg.)	42.9	0.0051	0.019	12.0	—	624.14	0.60	0.027
5 DP-FEC flows	30.0	0	0.00013	52.1	—	11	—	—
1 DP-FEC flow	60.0	1.1e-06	0.00084	40.9	—	22	—	—
1 DP-FEC flow	90.0	0.032	0.037	7.2	—	23	—	—
Total (Avg.)	42.9	0.0046	0.0055	44.09	—	14.29	0.49	0.030

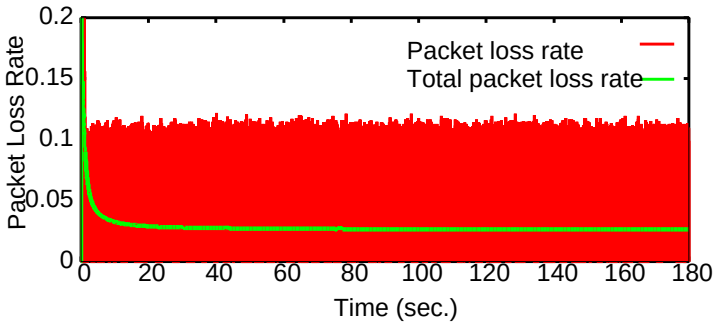
and relatively low average residal data loss rate, GENEVA is more appropriate for the Internet where high stability (i.e., high $TPindex$) is required.

5.3.3 Heterogeneous GENEVA Flows VS. TCP Flows in Single Bottleneck Link

To confirm the stability of heterogeneous GENEVA flows, we performed the experiments where 7 GENEVA flows or small/large Static-FEC flows or DP-FEC flows with different data transmission rates compete with long-lived TCP flows in the single bottleneck link. We set the number of GENEVA flows transmitting data packets at a rate of 30 Mbps to 5, and the other two flows have a rate of 60 Mbps and 90 Mbps respectively. The reason why we set 5 flows with a data rate of 30 Mbps is that GENEVA with a lower data rate tends to aggressively increase the $Fwnd$ and fluctuate network conditions. As shown in Table 5.2, the average number of bursty packet loss events of the GENEVA flows was suppressed to about 83.7, and the average data loss rate decreases to 0.1%. Whereas the $TPindex$ of GENEVA flows becomes almost the same as that of small/large Static-FEC flows, both the total average number of bursty packet loss events and the total residual



(a) Increased FEC window size



(b) Packet loss rate on the bottleneck link

Figure 5.11: **Trace of FEC window size of three selected GENEVA flows with different data transmission rates under RTT 10 ms, and packet loss rate on the bottleneck link.**

data loss rate of the GENEVA flows become lower than that of the small/large Static-FEC flows. In the case of DP-FEC flows, the total average residual data loss rate was not fully suppressed and become 0.46% by their aggressively increased $Fwnd$ (the total average $Fwnd$ of 44.0). In addition, the $TPindex$ of the DP-FEC flows decreases by more than 0.1, compared to that of the GENEVA flows. Fig. 5.11 shows the trace of the increased $Fwnd$ of the three selected GENEVA flows and the link loss rates. Although the GENEVA flow with a data rate of 30 Mbps increases the $Fwnd$ more aggressively, the link loss rates remain stable.

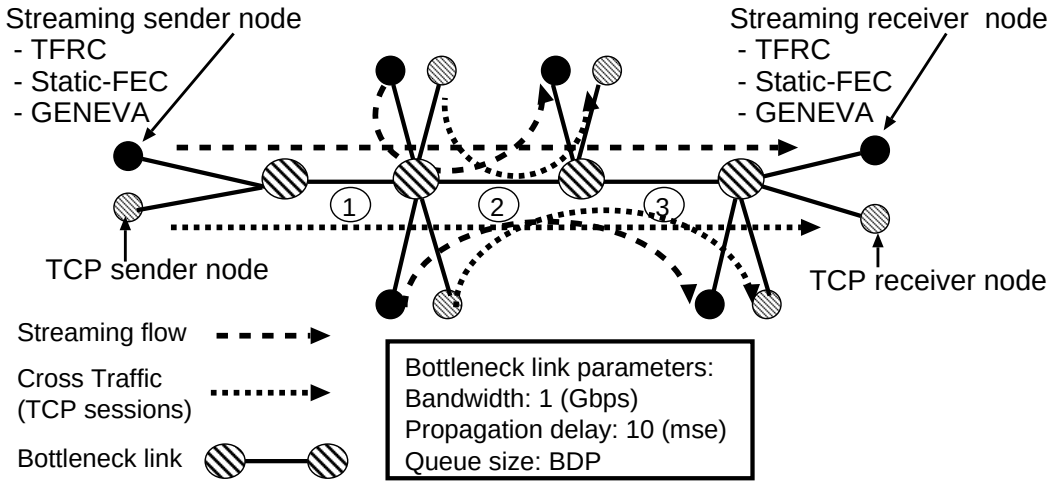
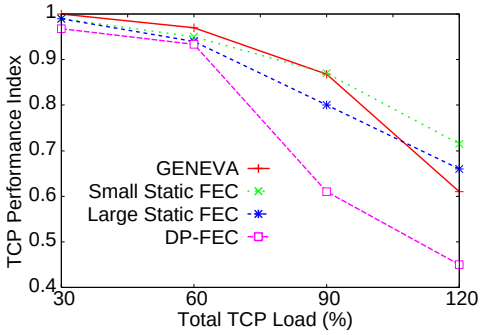


Figure 5.12: **The simulation topology (multiple bottleneck links).**

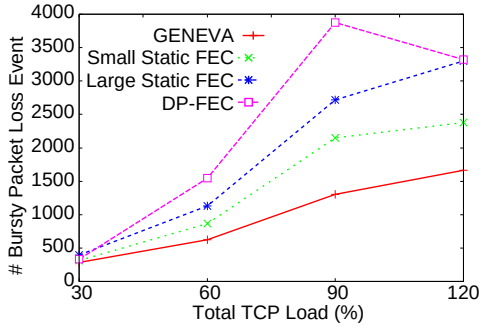
5.3.4 GENEVA Flows VS. TCP Flows in Multiple Bottleneck Links

We evaluated the GENEVA performance when competing with TCP flows in multiple bottleneck links. Fig. 5.12 shows the simulation topology where there are three bottleneck links. Each sender and receiver were connected to the bottleneck link through the 10 Gbps access link with propagation delay of about 1 ms. The numbers of both GENEVA and TCP sessions are 3. Each ρ_tcp in 3 TCP sessions was set to the same value, 10%/20%/30%/40%. The total ρ_tcp at the second bottleneck link thus becomes 30%/60%/90%/120%. All GENEVA or Static-FEC flows transmit data packets at a rate of 30 Mbps.

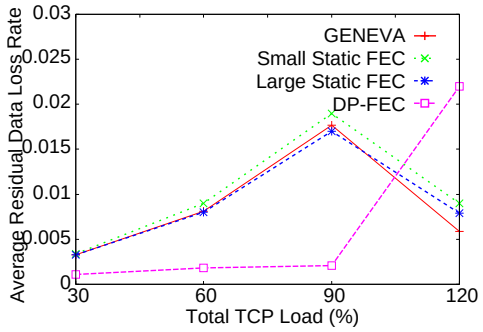
Fig. 5.13 shows the GENEVA performance metrics and the average FEC redundancy ratio under the total TCP load from 30% to 120%. We can see that, in 30% and 60% of the total TCP load at the second bottleneck link, the average FEC redundancy ratio is less than or almost equal to that of the small Static-FEC flows as shown in Fig. 5.13(d). The $TPindex$ of the GENEVA flows under that range of the total TCP load thus becomes almost the same value as that of the small Static-FEC flows, and higher value than that of the large Static-FEC flows, as shown in Fig. 5.13(a). Nevertheless, the average residual data loss rate of the GENEVA flows becomes lower than that of the small Static-FEC flows, and be-



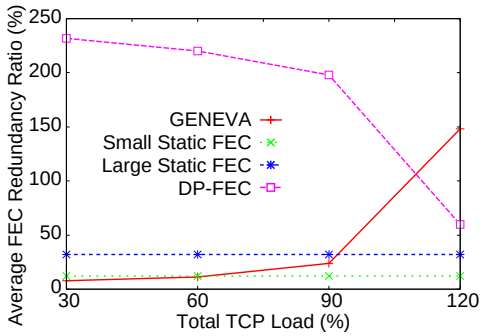
(a) TCP performance index



(b) The number of bursty packet loss event



(c) Average residual data loss rate



(d) Average FEC redundancy ratio (%)

Figure 5.13: **GENEVA performances under the total TCP from 30% to 120%.**

comes almost the same with that of the large static FEC-flows. This is because the number of bursty packet losses is suppressed as shown in Fig 5.13(b). This result indicates that the adjustment of the $Fwnd$ of the GENEVA flows works well to effectively utilize network resources without a severe adverse impact on performance of TCP flows. For the same reason, the $TPindex$ of the GENEVA flows in 90% of the total TCP load is almost the same with that of the small static FEC flows, and the average residual data loss rate of the GENEVA flows is almost the same with that of the large static FEC flows. Compared to the performance of the DP-FEC flows under that range of the total TCP load, the average FEC redundancy ratio of the DP-FEC flows becomes much higher than that of the GENEVA flows. This is because DP-FEC flows aggressively increase their $Fwnd$ to make a packet loss tolerance, which causes more bursty packet loss events. Although the average residual data loss rate of the DP-FEC flows becomes lower than that of the

DP-FEC flows, the DP-FEC $TPindex$ becomes lower than that of the GENEVA flows, especially in 90% of the total TCP load. Therefore, GENEVA flows under the total TCP load between 30% and 90% effectively adjust the $Fwnd$ to maintain higher $TPindex$ than those of both small Static-FEC and DP-FEC flows, and suppress residual data packet loss with smaller $Fwnd$ compared to large Static-FEC flows.

On the other hand, GENEVA flows in 120% of the total TCP load experience lower $TPindex$ compared to small and large Static-FEC flows. This is because GENEVA significantly increases the $Fwnd$ in a condition where the packet loss rate continues to be more than or equal to p_max. In such a situation (i.e., in high TCP load conditions), the $TPindex$ of GENEVA flows degrades. Although the average FEC redundancy ratio of DP-FEC flows becomes lower than that of the GENEVA flows, both the average residual data loss rate and $TPindex$ of the GENEVA flows become much better. This is because the $Fwnd$ adjustment with the supporting window control of the GNEVA flows, which especially have longer RTT, works well to achieve a stable network condition (i.e., DP-FEC flows with long RTT increase their $Fwnd$ without regard for RTT conditions). Thus, the GENEVA flows can retain higher performance than that of DP-FEC flows.

5.4 Summary

In this chapter, we proposed the GENEVA mechanism that achieves higher streaming quality for high-performance real-time streaming applications. To achieve high network utilization, the GENEVA mechanism allows the flow to maintain the moderate network congestion in which bursty packet loss event is suppressed to improve FEC recover capabilities through adjustment of the FEC window size. The rates of increase/decrease in the FEC window size using GMIAD algorithm is designed to minimize an adverse impact on competing TCP performance by effectively utilizing available bandwidth that TCP fails to copy, and to achieve stable conditions.

We verified the efficiency of GENEVA in comparison to the methods of DP-FEC, small Static-FEC and large Static-FEC in both single bottleneck link and multiple bottleneck links as simulation topology using an NS-2 simulator. As we can see the comparison results that Fig. 5.10 and 5.13 show, the TCP perfor-

mance index of GENEVA in all conditions becomes excellent. This is because the supporting window control of GENEVA can decide an acceptable degree of FEC redundancy according to network conditions so as to minimize an adverse effect of TCP performance. On the other hand, although the data loss rate of large Static-FEC becomes better than GENEVA in single bottleneck link, the corresponding TCP performance index becomes worse than GENEVA. The data loss rate of DP-FEC in low TCP load becomes better than GENEVA, and the corresponding TCP performance index becomes equal to GENEVA. However, in high TCP load, the TCP performance index of DP-FEC becomes worse than GENEVA due to an aggressive increase of the degree of FEC redundancy. In this context, GENEVA retains higher streaming quality while minimizing an adverse effect on TCP performance. In the next chapter, we present our conclusions and future directions.

Chapter 6

Conclusion

6.1 Summary

In this dissertation, motivated by the deployment of wide-area high-speed networks, we have proposed the two real-time streaming control mechanisms based on FEC, 1) the dynamic probing forward error correction (DP-FEC) and 2) GENEVA, the streaming control algorithm using generalized multiplicative-increase/additive-decrease (GMIAD). The proposed mechanisms aim to retain higher streaming quality while cooperating with TCP flows in view of the following current network situation:

- Due to the rapid increase in both access links and backbone networks, there is an increasing demand for high-quality and high-performance real-time streaming using high resolution such as HD or 4K and high frame rate. It is expected that network bandwidth usage of real-time streaming traffic will further grow as well as that of TCP traffic.
- TCP flows tend to transmit their packets in bursts especially in high-speed networks, and therefore may cause bursty packet losses. Such a bursty packet loss causes a significant degradation of streaming quality.
- The video streaming market such as IPTV has been receiving a lot of attention and growing rapidly, which produced an increasing requirement for maintaining higher streaming quality.

Current proposed congestion controllers fail to achieve higher streaming quality while effectively utilizing network resources in high-speed networks, where network congestion as indicated by packet loss occurs mainly due to TCP flows. Therefore, we focused on the scenario of multiple coexisting flows along a high-speed network path, where both high-performance real-time streaming flows and TCP flows are competing, and proposed the new congestion controllers that are more effective and suitable in the current network situation.

First of all, since we apply FEC to the proposed algorithms, we bring out FEC performance by using an actual high-performance real-time video streaming application and investigating three AL-FEC codes: 2D parity check codes, Reed-Solomon over GF(2^8) codes, and LDPC-Staircase codes, all of them being currently standardized within Internet Engineering Task Force (IETF) [34]. Based on the experimental results in terms of recovery capabilities, frame delay and processing load, we have identified the optimal configuration for a given incoming loss rate. More precisely, under low packet loss conditions, all FEC codes achieve high quality video playback since erasures are recovered within their real-time constraints. From the processing load point of view, RSE codes create an important CPU load compared to the other codes. Under high packet loss conditions, the 2D and RSE codes no longer perform well in terms of the recovery capability and real-time performance. On the opposite, the LDPC-Staircase codes using small source block length achieve almost optimal performance. The investigation results are of high importance, because practical results and evaluations as a function of the FEC codes have not been fully studied in the context of high-performance real-time applications.

DP-FEC can be utilized for the purposes of IPTV, e-learning or international symposiums that strongly require maintaining the best possible streaming quality. DP-FEC is designed as an end-to-end model in consideration of the following criteria:

1. Utilizing network resources effectively, the degree of FEC redundancy during data transmission is increased to make a packet loss tolerance as high as possible. At the same time, network estimation for congestion control can be conducted.

2. In accordance with the network estimation, DP-FEC adjusts the degree of

FEC redundancy to minimize the impact of FEC on performance of competing TCP flows.

DP-FEC provides a promising method for achieving higher streaming quality while seeking the behaviors of competing TCP flows. DP-FEC estimates the network conditions by dynamically adjusting the degree of FEC redundancy while trying to recover lost data packets. By successively observing variation in the intervals between packet loss events, DP-FEC effectively utilizes network resources. It aggressively increases the degree of FEC redundancy to make a packet loss tolerance as high as possible while minimizing the performance impact of competing TCP flows. We verified the efficiency of DP-FEC using an NS-2 simulator. The main results show that except in the case where TCP load becomes extremely high, DP-FEC suppresses data loss rate like large Static-FEC with 80% FEC redundancy, and maintains TCP performance almost as high as small Static-FEC with 10% FEC redundancy. In this context, DP-FEC enables high-performance streaming flows to utilize network resources effectively by using FEC and maintain high streaming quality while cooperating with TCP flows.

GENEVA also provides a promising method for achieving higher streaming quality by adding redundant data packets while effectively utilizing network resources. Assuming an usage environment like the Internet where real-time streaming flows with various data transmission rates and RTTs compete and network stability is strongly required, GENEVA is designed as an end-to-end model using the following criteria:

1. To achieve high network utilization, GENEVA allows a high-performance streaming flow to maintain moderate network congestion, in which packet loss rate is less than p_{max} (a predefined constant value).

2. An GENEVA flow should converge at an appropriate equilibrium point to recover data packet losses under stable conditions during transmission. To achieve such a condition, supporting window control is needed to specify an acceptable degree of FEC redundancy.

3. To suppress an occurrence of bursty packet loss that TCP flows cause and improve FEC recover capabilities, the degree of FEC redundancy is increased to prevent their congestion window sizes from increasing considerably. At the

same time, the increase in FEC redundancy should not interact badly with competing TCP flows.

GENEVA combats bursty packet losses through adjustment of the transmission rate using the GMIAD mechanism. GENEVA avoids low-achieving throughput by allowing the streaming flows to maintain moderate network congestion, and also considers the effect on performance of other competing TCP flows. The rate of increase/decrease in the degree of FEC redundancy using GMIAD algorithm is designed to minimize an adverse impact on competing TCP performance by effectively utilizing available bandwidth that TCP fails to copy, and to achieve stable conditions. We verified the efficiency of GENEVA using an NS-2 simulator. The main results show that GENEVA suppresses data loss rate like large Static-FEC with 30% FEC redundancy, and maintains TCP performance almost as high as small Static-FEC with 10% FEC redundancy. In comparison with DP-FEC, under a high TCP load condition, GENEVA achieves TCP performance 30% higher than DP-FEC and lower data loss rate due to the supporting window control. In this context, GENEVA retains high streaming quality while minimizing an adverse effect on TCP performance under stable network conditions.

The proposed mechanisms are more suitable for high-performance real-time streaming applications in an environment where both network connection speed and IP traffic have been growing. They can solve the problems that existing congestion controllers cannot address, and provide more practical and promising solutions befitting the current network situation and application requirement for streaming quality according to user usage environments. This dissertation can contribute largely to further growth and promotion of high-performance real-time streaming which is subject to an increasing demand for higher streaming quality.

6.2 Future Directions

In our future work, we will make deeper analysis of the DP-FEC and GENEVA algorithm to optimally adjust the parameters (e.g., *Threshold FEC Impact* in DP-FEC and the scaling properties of the total window size in GENEVA) for ever-changing network conditions. We will then implement an actual high-performance application equipped with both DP-FEC and GENEVA, and evaluate both the DP-

FEC and GENEVA mechanism competing with many high-performance streaming and TCP flows in high speed networks. This evaluation will further improve both algorithms. In addition, we will start to standardize the proposed mechanisms in IETF. The standardization activity will accelerate the deployment of high-performance real-time streaming applications over high-speed networks.

With the advent of software defined network (SDN), network administrators come to be able to have programmable central control of network traffic and operate flexibly managed network. In the near future, the growth of SDN will further promote the video streaming market in terms of ensuring available bandwidth for competing streaming flows, where the requirement for maintaining streaming quality will become more significant. Since DP-FEC can be applied in such a managed network in order to maintain the best possible streaming quality, we will investigate and strengthen the effectiveness of DP-FEC with consideration for functions of managed networks.

Since the proposed mechanisms inherently consist of network estimation and corresponding adjustment of the transmission rate on an end-to-end model, there is a possibility that the ideas and concepts could be applied to other protocols. For instance, although we focused on high-performance real-time streaming applications adopting UDP, the main ideas of the proposed mechanisms could be applied to TCP so as to improve TCP performance. Based on this research knowledge, we will make TCP work well for various real-time streaming applications including Video on Demand (VoD) in both wire and wireless networks. This work will contribute to further improvement of streaming quality in future network environments where a large number of various real-time streaming flows have been promoted.

Bibliography

[1] Akamai Technologies, Inc. 3rd Quarter, 2011 The State of the Internet, available from *http://www.akamai.com/stateoftheinternet/*.

[2] Cisco white paper, Forecast and Methodology, 2010-2015, available from *http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html/*.

[3] 2011-2012 Report of the ICFA-SCIC Monitoring Working Group, available from *http://www-iepm.slac.stanford.edu/pinger/*.

[4] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the Internet," *Proc. IEEE INFOCOM'99*, New York, USA, Mar. 1999.

[5] N. Nakashima, K. Okamura, J.S. Hahm, Y.W. Kim, H. Mizushima, H. Tatsumi, B.I. Moon, H.S. Han, Y.J. Park, J.H. Lee, S.K. Youm, C.H. Kang, and S. Shimizu, "Telemedicine with digital video transport system in Asia -Pacific area," *Proc. the 19th International Conference on Advanced Information Networking and Applications*, Mar. 2005.

[6] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 3550, July 2003.

[7] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *Proc. ACM Comput. Commun*, Vol.27, No.2, pp.24–36, Apr. 1997.

[8] T. Tsugawa, N. Fujita, T. Hama, H. Shimonishi and T. Murase, "TCP-AFEC: An adaptive FEC code control for end-to-end bandwidth guarantee," *Proc. International Packet Video Workshop* (*PV 2007*), Nov. 2007.

[9] A. Ogawa, K. Kobayashi, K. Sugiura, O. Nakamura, and J. Murai, "Design and implementation of DV based video over RTP," *Proc. International Packet Video Workshop* (*PV 2000*), May 2000.

[10] E. Biersack. "Performance evaluation of forward error correction in ATM networks," *Proc. ACM SIGCOMM'92*, Aug. 1992.

[11] N. Shacham and P. Mckenney, "Packet recovery in high-speed networks using coding and buffer management," *Proc. IEEE INFOCOM'90*, San Francisco, CA, June 1990.

[12] Y. Xunqi, J.W. Modestino, R. Kurceren, and Y.S. Chan, "A model-based approach to evaluation of the efficacy of FEC coding in combating network packet losses," *IEEE/ACM Trans. on Networking*, Vol.16, No.3, pp.628–641, June 2008.

[13] I. Cidon, A. Khamisy, and M. Sidi, "Analysis of packet loss processes in high-speed networks," *IEEE Trans. Inf. Theory*, Vol. 39, No. 1, pp. 98–108, Jan. 1993.

[14] K. Matsuzono, J. Detchart, M. Cunche, V. Roca, and H. Asaeda, "Performance analysis of a high-performance real-time application with several AL-FEC schemes," *Proc. IEEE Local Computer Networks* (*LCN*), Oct. 2010.

[15] J. Lacan, V. Roca, J. Peltotalo, and S. Peltotalo, "Reed-Solomon Forward Error Correction (FEC) Schemes," IETF RFC 5510, Apr. 2009.

[16] V. Roca, M. Cunche, J. Lacan, A. Bouabdallah and K. Matsuzono, "Simple Reed-Solomon Forward Error Correction (FEC) Scheme for FECFRAME," IETF Internet Draft, draft-ietf-fecframe-simple-rs-06, Jan. 2013.

[17] V. Roca, C. Neumann and D. Furodet, "Low Density Parity Check (LDPC) Staircase and Triangle Forward Error Correction (FEC) Schemes," IETF RFC 5170, June 2008.

[18] V. Roca, M. Cunche and J. Lacan, "Simple Low-Density Parity Check (LDPC) Staircase Forward Error Correction (FEC) Scheme for FECFRAME," IETF RFC 6816, Dec. 2012.

[19] J. S. Ahn, S. W. Hong, and J. Heidemann, "An adaptive FEC code control algorithm for mobile wireless sensor networks," *Journal of Communications and Networks*, Vol. 7, No. 4, pp. 489–498, Dec. 2005.

[20] C. Neumann, V. Roca, A. Francillon, and D. Furodet, "Impacts of packet scheduling and packet loss distribution on FEC performances: observations and recommendations," *Proc. CoNEXT'05*, Toulouse, France, Oct. 2005.

[21] C. Lamoriniere, A. Nafaa, and L. Murphy "Dynamic switching between adaptive FEC protocols for reliable multi-source streaming," *Proc. IEEE Globecom'09*, Hawai, USA, Nov. 2009.

[22] M. Cunche and V. Roca, "Optimizing the error recovery capabilities of LDPC-staircase codes featuring a Gaussian elimination decoding scheme," *Proc. IEEE International Workshop on Signal Processing for Space Communications* (*SPSC'08*). Oct. 2008.

[23] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "The Use of Forward Error Correction (FEC) in Reliable Multicast," IETF RFC 3453, Dec. 2002.

[24] D. MacKay, "Information Theory, Inference and Learning Algorithms," Cambride University Press, ISBN: 0521642981, 2003.

[25] M. Cunche and V. Roca, "Improving the ecoding of LDPC codes for the packet erasure channel with a hybrid Zyablov iterative decoding/Faussian elimination scheme," Research Report 6473, INRIA, Mar. 2008.

[26] V. Zyablov and M. Pinsker, "Decoding complexity of low-density codes for transmission in a channel with erasures," Translated from Problemy Peredachi Informatsii, Jan. 1974.

[27] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann. "RTP Payload Format for DV (IEC 61834) Video," IETF RFC 3189, Jan. 2002.

[28] OpenFEC.org: because open, free AL-FEC codes and codecs matter, available from *http://openfec.org*.

[29] S. Floyd, "Congestion Control Principles," IETF RFC 2914, Sep. 2000.

[30] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Trans. on Networking*, Vol.7, No.4, pp.458–472, Aug. 1999.

[31] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *Proc. ACM Comput. Commun*, Vol.30, No.4, pp.43–56, Oct. 2000.

[32] P. Papadimitriou and V. Tsaoussidis, "A rate control scheme for adaptive video streaming over the Internet," *Proc. IEEE ICC'07*, Glasgow, Scotland, June 2007.

[33] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," IETF RFC 5348, Sep. 2008.

[34] The Internet Engineering Task Force, available from *http://www.ietf.org/*.

[35] Pro-MPEG Forum, available from *http://pro-mpeg.org/*.

[36] Tom Kelly, "Scalable TCP: Improving performance in high-speed wide area networks," *PFLDnet Workshop*, Feb. 2004.

[37] S. Floyd, "Highspeed TCP for Large Congestion Windows," IETF RFC 3649, Dec. 2003.

[38] HighSpeed TCP Web Page, available from *http://www.icir.org/floyd/hstcp.html*.

[39] S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," *PFLDnet Workshop*, Feb. 2004.

[40] R. S. Prasad, M. Jain, and C. Dovrolis, "Delay-based congestion avoidance for TCP," *IEEE/ACM Trans. on Networking*, Vol. 11, No. 3, pp. 356–369, June 2003.

[41] H. Seferoglu, U.C. Kozat, M.R. Civanlar, and J. Kempf, "Congestion state-based dynamic FEC algorithm for media friendly transport layer," *Proc. International Packet Video Workshop* (*PV 2009*), May 2009.

[42] H. Seferoglu, A. Markopoulou, U.C. Kozat, M.R. Civanlar, and J. Kempf, "Dynamic FEC algorithms for TFRC flows," *IEEE Trans. on Multimedia*, Vol. 12, No. 8, pp. 869–885, Dec. 2010.

[43] J. R. Taal, K. Langendoen, A. van der Schaaf, H. van Dijk, and R. Lagendijk, "Adaptive end-to-end optimization of mobile video streaming using QoS negotiation," *Proc. IEEE ISCAS*, Scottsdale, AZ, May 2002.

[44] S. S. Karande and H. Radha, "Rate-constrained adaptive FEC for video over erasure channels with memory," *Proc. IEEE ICIP*, Hainan, China, May 2008.

[45] V. Paxson, "End-to-End Internet packet dynamics," *Proc. IEEE/ACM Trans. on Networking*, Vol. 7, No. 3, pp.277–292, June 1999.

[46] J.C. Bolot, H. Crépin, and A. Vega-Garcia, "Analysis of audio packet loss over packet-switched networks," *Proc. ACM NOSSDAV'95*, New Hampshire, USA, Apr. 1995.

[47] H. Wu, M. Claypool, and R. Kinicki, "Adjusting forward error correction with quality scaling for streaming MPEG," *Proc. ACM NOSSDAV'05*, Washington, USA, June 2005.

[48] C. Bao, X. Li, and J. Jiang, "Scalable application-specific measurement framework for high performance network video," *Proc. ACM NOSSDAV'07*, Illinois, USA, June 2007.

[49] Y. Zhang and D. Loguinov, "Oscillations and buffer overflows in video streaming under non-negligible queuing delay," *Proc. ACM NOSSDAV'04*, Cork, Ireland, June 2004.

[50] S. Zeadally, H. Moustafa, "Internet protocol television (IPTV): Architecture, Trends, and Challenges," *Proc. IEEE Systems Journal*, Vol. 5, No. 4, pp. 528–527, Dec. 2011.

[51] E. Altman, C. Barakat, and V.M. Ramos, "Queuing analysis of simple FEC schemes for IP telephony," *Proc. IEEE INFOCOM'01*, Nov. 2001.

[52] C. Marcondes, A. Persson, M.Y. Sanadidi, M. Gerla, H. Shimonishi, T. Hama, and T. Murase, "Inline path characteristic estimation to improve TCP performance in high bandwidth-delay networks," *PFLDnet Workshop*, Feb. 2006.

[53] M. Zhan, Z. Dongsheng, andL. Jihong, "A survey of optical networks and its visual management and control," *Proc. Advanced Computer Control (ICACC)*, Harbin, China, Jan. 2011.

[54] T. Kondo, K. Nishimura and R. Aibara, "An efficient FEC method for high-quality video transmission on the broadband internet," *IEICE Trans. on Communications*, Vol.E87-B, No.3, pp. 643-650, Mar. 2004.

[55] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," *Proc. ACM SIGCOMM'06*, Piza, Italy, Sep. 2006.

[56] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," IETF RFC 4340, Mar. 2006.

[57] Y. Gu, X. Hong, and R. L. Grossman, "Experiences in design and implementation of a high performance transport protocol" *Proc. ACM/IEEE Supercomputing 2004* (*SC2004*), Pittsburgh, PA, USA, Nov. 2004.

[58] B. Ngamwongwattana and A. Sombun, "On cross-induced congestion in media streaming," *Proc. Communications and Information Technologies* (*ISCIT*), Oct. 2008.

[59] J. Feng and L. Xu, "TCP-Friendly CBR-Like rate control," *Proc. IEEE ICNP*, Oct. 2008.

[60] K. Matsuzono, H. Asaeda, and J. Murai, "DP-FEC: Dynamic probing FEC for high-performance real-time interactive video streaming," *Journal of Information Processing, IPSJ*, Vol. 20, No. 1, pp. 185–195, Jan. 2012.

[61] K. Matsuzono, H. Asaeda, O. Nakamura, and J. Murai, "GENEVA: Streaming control algorithm using generalized multiplicative-increase/additive-

decrease," *Journal of Information Processing, IPSJ*, Vol. 21, No. 1, pp. 109–121, Jan. 2013.

[62] H. Jiang and C. Dovrolis, "Why is the Internet traffic bursty in short time scales ?," *Proc. ACM SIGMETRICS'05*, June 2005.

[63] R. S. Prasad, C. Dovrolis and M. Thottan, "Router Buffer Sizing Revisited: The role of the output/input capacity ratio," *Proc. ACM CoNEXT'07*, Dec. 2007.

[64] M. Mathis, J. Semke, J. Mahdavi, T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Proc. ACM Comput. Commun*, Vol. 27, No. 3, pp. 67–82, July 1997.

[65] G. Raina, D. Towsley, and D. Wischik, "Part II: control theory for buffer sizing," *Proc. ACM Comput. Commun*, Vol. 35, No. 3, pp. 79–82, July 2005.

[66] K. Matsuzono, K. Sugiura, and H. Asaeda, "Adaptive rate control with dynamic FEC for real-time DV streaming," *Proc. IEEE Globecom'08*, New Orleans, USA, Dec. 2008.

[67] J. Postel, "Transmission Control Protocol," IETF RFC 793, Sep. 1981.

[68] J. Postel, "User Datagram Protocol," IETF RFC 768, Aug. 1980.

[69] V. Jacobson, "Congestion avoidance and control," *Proc. ACM SIGCOMM'88*, Nov. 1988.

[70] D. Chiu and R.Jain, "Analysis of the increase and decrease algorithm for congestion avoidance in computer networks," *Comput. Netw. ISDN Syst. J.*, Vol.17, No.1, pp.1–14, June 1989.

[71] L. Rizzo, dummynet, available from *http://info.iet.unipi.it/l̃uigi/dummynet/*.

[72] X. WANG and H. Schulzrinne, "Comparison of adaptive internet multimedia applications," *IEICE Trans. on Communications*, Vol. E82-B, No. 6, pp. 806–818, June 1999.

[73] K. Matsuzono, H. Asaeda, K.Sugiura, O. Nakamura, and J. Murai, "Analysis of FEC function for real-time DV streaming," *Proc. Asian Internet Engineering Conference* (*AINTEC'07*), LNCS 4866, Nov. 2007.

[74] Digital Video Transport System (DVTS), available from *http://www.sfc.wide.ad.jp/DVTS/*.

[75] Nishimura. K, Kondo. T, and Aibara. R, "An MPEG2 over IP transfer system integrating live distance lecture and live distance learning Archive," *Proc. International Conference on Computers in Education* (*ICCE'02*), Auckland, New Zealand, Dec. 2002.

[76] J.C. Bolot, S. Fosse-Parisis, and D. Towsley, "Adaptive FEC-based error control for Internet telephony," *Proc. IEEE INFOCOM'99*, New York, USA, Mar. 1999.

[77] M. Watson, A. Begen, V. Roca, "Forward Error Correction (FEC) Framework," IETF RFC 6363, Oct. 2011.

[78] The Network Simulator, NS-2, available from *http://www.isi.edu/nsnam/ns/*.

[79] L. Kleinrock, "Queuing Theory," Wiley, New York, 1975.

[80] J.C. Bolot and A. V. Garcia, "Control mechanisms for packet audio in the Internet," *Proc. IEEE INFOCOM'96*, Mar. 1996.

[81] C. Perkins and O. Hodson, "Options for Repair of Streaming Media," IETF RFC 2354, June 1998.

[82] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource reservation protocol," *Proc. IEEE Network*, Sep. 1993.

[83] R. Braden, Ed., L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification," IETF RFC 2205, Sep. 1997.

[84] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. on Networking*, Vol.3, No.4, pp.365–386, Aug. 1995.

[85] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A simple model and its empirical validation," *Proc. ACM SIGCOMM'98*, Sep. 1998.

Appendix A

Adaptive Rate Control with Dynamic FEC for Real-Time DV Streaming

For higher streaming quality, a data sender adjusts the data transmission rate according to the network condition between the sender and receiver. The sender and the receiver exchange information about the network condition to decide the appropriate transmission rate. However, monitoring each data flow in real time is difficult, and controlling the sender to adjust the best quality for each receiver in a heterogeneous environment is a real challenge. We study packet loss patterns and FEC recovery rate upon data transmission in a congested network, and define an adaptive rate control mechanism that dynamically adjusts the transmission rate and the FEC encoding rate. We then show our adaptive DV Transmission System (DVTS) that supports appropriate rate control for provisioning multimedia streaming with the best possible quality, and evaluate the system on top of our testbed network.

A.1 Introduction

The Digital Video (DV) format gives consumers the ability to edit video with almost professional quality without adding the special hardware requirements to consumer electronics or PCs. Many consumer products or equipments support the DV format, helping it fulfill its promise. Due to dissemination of high speed DSL and FTTH, high quality DV streaming over IP will become common on the Internet.

On the other hand, requirements for receiving high quality stream that consumes a large amount of bandwidth and competes against other data flows are not always fulfilled to everyone. Real-time multimedia stream in the Internet is usually transmitted over RTP [6] carried on top of IP and UDP, and RTCP [6] sent with RTP supports "rate control" as one of the fundamental functions to adjust the transmission rate based on the capacity of the available bandwidth or to adapt to network congestion. To make things work properly, precise knowledge of the available bandwidth between a sender and a receiver is necessary, but it is difficult to expect appropriate data transmission rate as the network condition for each user is very changeable in the heterogeneous environment.

To provide stable streaming quality, in a different technological approach, a sender adds redundant data to its stream and a receiver detects and corrects errors occurring during transmission without the need to ask the sender for additional data. A "Forward Error Correction (FEC)" [80, 81] is the well-known algorithm that has been notably used in various applications. Although the FEC encoding rate can be coordinated by a sender and a receiver, it is difficult to decide the most appropriate encoding rate in real time.

Accordingly, we study a mechanism that *simultaneously* uses both rate control and FEC technologies for providing smooth and high quality end-to-end streaming. The mechanism dynamically adjusts transmission rate and FEC rate by estimating available bandwidth between the sender and receiver. One of the difficulties in this mechanism is that when FEC rate is increased to recover packet loss, the total amount of transmitted data is also increased in the network path and additional packet loss may occur as a result. Since the effective rate control may work against the optimal FEC encoding rate, knowing a good balance to decide the timing and the redundancy level for increasing or decreasing the transmission rate and FEC

rate is vital.

We propose and show an *adaptive* rate control mechanism for smooth DV streaming. We study packet loss patterns and FEC recovery rate by simulation, and define an effective algorithm that incorporates transmission rate control with adjustment of the FEC encoding rate. The bandwidth estimation is calculated from the inputs given by the condition of consecutive packet loss and FEC recovery rate. We then show an actual implementation of adaptive DV Transmission System over IP (DVTS) [9, 27] that gives good performance for provisioning multimedia streaming with the adaptive rate control mechanism.

A.2 Quality Adaptation for Streaming over IP

A.2.1 Problem Statements

Audio and video transmission encoded for real-time streaming generally tries to keep pace with the user's communication condition. For maintaining streaming quality, an application adopts a resource reservation or congestion control mechanism to avoid reductions in streaming quality caused by packet loss, jitter, or delay. To escape from interruptions and stalling, an application compensates packet losses by various approaches whose suitability may be dependent on encoding techniques and communication environments.

Network resource management is the technology being used to keep streaming quality. One of its major protocols is "Resource Reservation Protocol (RSVP)" [82, 83], which reserves network bandwidth between a sender and a receiver for data transmission. "Class-Based Queueing (CBQ)" [84] is a resource sharing mechanism that shares the bandwidth on a link in packet networks. Both require resource management mechanisms at the network equipment level, where requiring a gateway to accommodate an essential component is hard to assume and lacks the flexibility of communication. *It is difficult to adopt these techniques for every common streaming architecture used over the wide-spread Internet.*

A congestion control mechanism would be indispensable to avoid packet loss and make end-to-end DV streaming with higher quality workable. One of the possible approaches is "TCP-friendly rate control" [30, 31, 33]; it behaves fairly

with respect to coexistent TCP flows, in order to provide a promising mechanism for avoiding severe fluctuations in the transmission rate. While it ensures fairness with competing TCP flows, the throughput of non-TCP flows does not exceed the throughput of a conformant TCP connection under the same conditions, but *this condition is not reasonable for DV streaming that consumes bandwidth* (e.g. 30Mbps).

As an open-loop approach, Forward Error Correction (FEC) is effective especially for streaming applications because it adds redundant information to packets in order to allow a receiver to correct missing packets without retransmission requests. The redundancy level is defined as FEC encoding rate, which is decided by a Bit Error Rate (BER) at the receiving side and the previously used encoding rate. FEC rate control is used to change the redundancy of data; a higher value increases the possibility of recovering the stream but increases the amount of traffic. Thus, *simply increasing FEC rate without intellectual decision does not improve the streaming quality, because additional network congestion may be occurred.*

A.2.2 Design Aspects

Although designing the ideal transmission rate control mechanism is necessary for streaming applications, there are various difficulties to find the best transmission rate to maximize streaming quality on dynamic networks. Even though the best transmission rate is examined, smooth rate control that minimizes lost packets and jitter is always preferred with respect to abrupt quality changes in a heterogeneous communication environment. The requirement is sensitive especially for transmitting DV stream over IP, since DV stream consumes a large amount of bandwidth like 30 Mbps generally.

As part of a common streaming application, RTCP is used to estimate the available bandwidth between a sender and a receiver and notify the network condition to the sender. After the data transmission rate is negotiated whether it is increased or decreased, the application triggers rate control to adjust data transmission rate being suitable for the available network bandwidth.

[45] studied Internet packet dynamics. Under the packet loss conditions the author investigated, the bottleneck comes from the slowest forwarding element (i.e. router or link) in the end-to-end chain that comprises the network path. On

the other hand, it is complex to distinguish and expect packet loss patterns only by observing the current data packet loss. According to these thoughts, although "packet loss rate" becomes an important factor for expecting network condition, getting and using other characteristics would be also encouraged to adapt usable bandwidth to streaming applications.

In [46], the authors considered the problem of distributing real-time data and measured audio packets and packet loss patterns. They analyzed that the number of consecutively lost packets is small especially when the network load is low or moderate. Based on their results, only about 10% of the total lost packets are in consecutive losses of more than three packets in the stream. Therefore, we define "consecutive packet loss" as loss of more than three packets consecutively, and assume "the number of consecutively lost packets" is a useful indicator for the condition of the network and triggering transmission rate control.

Additional analysis in [46] relates to packet loss recovery. An FEC mechanism is effective for a streaming application especially when lost packets are dispersed through the stream of packets, or when network condition is unstable or changed at frequent intervals. This analysis inspires us to monitor the "FEC recovery rate" as one network condition value, because the packet loss is recovered only when it is lower than the FEC encoding rate. For instance, if FEC does not completely recover the lost packets during streaming for a certain period, it implies that the network congestion may not be converged and more rate control would be needed.

A.3 Adaptive Rate Control

A.3.1 Overview

We propose an *adaptive* rate control mechanism that seamlessly cooperates with "transmission rate control" and "dynamic FEC" technologies for providing the best possible DV streaming quality. The mechanism investigates three characteristics, "packet loss rate", "the number of consecutively lost packets", and "FEC recovery rate", during streaming, to characterize the network condition between a sender and a receiver. It then decides the appropriate "flow type", which is defined as a combination of "data transmission rate" and "FEC encoding rate", and

dynamically adjusts these rates based on the network condition. As the concrete implementation, we implemented "adaptive DVTS". It supports that a receiver gives the information of "packet loss rate", "the number of consecutively lost packets", and "FEC recovery rate" to a sender by extended RTCP RR, and a sender adjusts the transmission rate and the FEC encoding rate of the stream. Regarding an FEC codec, we used the Reed-Solomon FEC codec developed by L. Rizzo.

A.3.2 Simulation

In order to know how the flow type can be effectively adjusted during streaming, we analyzed the correlation between packet loss patterns and FEC recovery rate by using NS-2 [78]. The parameters and procedures used in the simulation are defined as follows;

1. A sender transmits T Mbps UDP packets, and a receiver receives the packets and measures; 1) the packet loss rate ($P_{los}\%$), 2) the number of consecutively lost packets (N), and 3) the number of non-recovered UDP packets (L) within 5 seconds. Here, a combination of packet loss rate and the number of consecutively lost packets is defined as a "packet loss pattern" (P_{los}, N).

2. In the presence of competitive traffic in the simulation network, the receiver measures the different packet loss pattern and the number of non-recovered UDP packets within 5 seconds. This measurement is repeated for each 5 seconds by changing competitive traffic patterns.

3. In the same network condition in which L was given, the sender transmits T Mbps UDP packets with $F_{enc}\%$ FEC encoding rate packets, and the receiver measures the number of non-recovered UDP packets (L') within 5 seconds. This measurement is also repeated.

4. "FEC recovery rate" ($F_{rec}\%$) defined with the following form is calculated for each flow type (T, F_{enc})

$$F_{rec} = \begin{cases} \frac{L-L'}{L} \cdot 100, & (L > L', L \neq 0) \\ 0. & (L \leq L', L \neq 0) \end{cases} \tag{A.1}$$

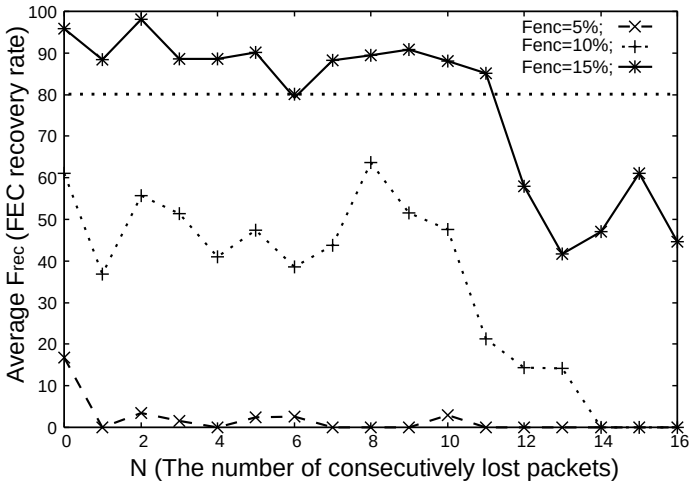


Figure A.1: **The number of consecutively lost packet and FEC recovery rate.**

As one of the experimental results (where $T = 30$), the correlation between the packet loss pattern (where $P_{los} = 1$ to 3%) and the average F_{rec} with different F_{enc} is shown in Fig. A.1. The result indicates that the packet loss pattern is a good metric to change the flow type. When we assume that F_{rec} should be always higher than 80% for the best possible streaming, a sender should add 15% F_{enc} with the same transmission rate when N is below 11 in $P_{los} = [1, 3]$. On the other hand, when N is more than 11, the FEC effectiveness deteriorates severely, and hence the sender should either add more than 15% F_{enc} with the same transmission rate or reduce the data transmission rate. If the UDP packets with higher FEC encoding rate than 15% provides 80% or higher F_{rec}, the sender adds more FEC redundancy (e.g. 20%). If the UDP packets does not provide 80% or higher F_{rec}, the sender reduces the transmission rate to fulfill the 80% or higher F_{rec} quality.

A.3.3 Experimental Analysis

In this section, we create an algorithm that provides reference of the flow types and patterns of changing the transmission state by using DVTS implementation [73], and propose the *adaptive* DVTS that supports the defined algorithm. Table A.1 shows the hardware spec and OS of a DVTS sender, a DVTS receiver, and a *dummynet* [71] network emulator. The sender and the receiver directly connect to

Table A.1: **Hardware in our experiment.**

	sender and receiver	*dummynet*
CPU	Intel Pentium M 1GHz	Intel Xeon 3.60GHz
Memory	512 MB	3 GB
OS	Linux Kernel 2.6.17	FreeBSD 5.4 Release
NIC	RealTech 100Base-TX	Intel 1000Base-T

the *dummynet* that proportionally narrows the available bandwidth from 50 Mbps to 25 Mbps within 125 seconds for each test.

Effectiveness of frequent transmission rate changes

Although adjusting the transmission rate in streaming is necessary for quality adaptation, according to the property of DV transmission, frequently changing the data transmission rate by discarding frame data is not highly effective or rather decreases DV streaming quality [72]. In our experiment, we also observed that the DV streaming quality was not effectively improved by changing the transmission rate frequently. Therefore, we decided that the adaptive DVTS does not change the transmission rate frequently or susceptibly, but simply changes the rate from 100% to 50% or vice versa.

Correlation between FEC encoding rate and non-recovery rate in full rate DV stream

In the next experiment, a DV sender sent full rate DV stream (30 Mbps) to a receiver and statically added FEC redundancy in the streaming data. The FEC encoding rate was changed from 0% to 10%, 20% and 30% step-by-step. Fig.A.2 shows the measurement results. In the regular DV transmission (i.e. with no FEC redundancy), the packet loss was observed from 50 seconds (at 40 Mbps bandwidth). In the DV transmission with 10% FEC redundancy, the packets were lost in 65 sec. (37 Mbps b/w). With 20% FEC redundancy, non-recovered data was observed in 75 sec. (35 Mbps b/w). With 30% FEC redundancy, non-recovered data was observed in 85 sec. (33 Mbps b/w). In 90 sec. from the transmission (32 Mbps b/w), the non-recovery rate was turned over by the transmission with higher FEC redundancy, because the total amount of traffic was gradually increased and

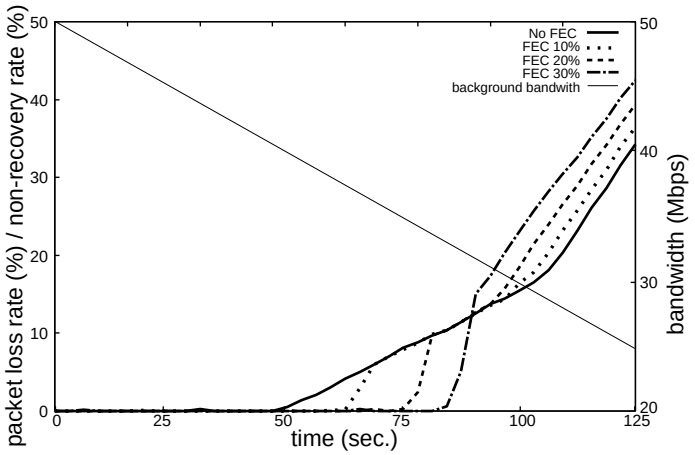


Figure A.2: **Packet loss and non-recovery rate in full rate DVTS.**

hence additional packet loss was triggered. In addition, the non-recovered data with more than 40% FEC was observed in nearly the same timing in 30% FEC redundancy. Therefore, we decided that maximum FEC rate in the full rate should be 30%.

Correlation between FEC encoding rate and non-recovery rate in half rate DV stream

Next, the sender changed the DV transmission rate to half (15 Mbps) and the FEC encoding rate from 50% up to 100%. Fig.A.3 shows the results that when the sender sent half rate DV stream with no FEC redundancy, the packet loss was observed in nearly the same timing in the full rate transmission (i.e. in 50 seconds)[1]. However, in the half rate transmission with 50% FEC redundancy, non-recovery data was appeared in 88 sec. (32 Mbps b/w), and with 60% FEC redundancy, non-recovery data was observed in 95 sec. (31 Mbps b/w). With 100% FEC redundancy, DV packets would be able to be recovered if the available bandwidth is more than 26 Mbps. According to these results, the transmission rate should be reduced from full to half rate with more than 50% FEC rate to give higher effort when network condition is not good to keep the full rate transmission. Moreover, because adding more than 110% FEC redundancy was not highly effective than 100% FEC

[1]This happened due to the current Linux driver that bursty transmits DV frames in a particular timing.

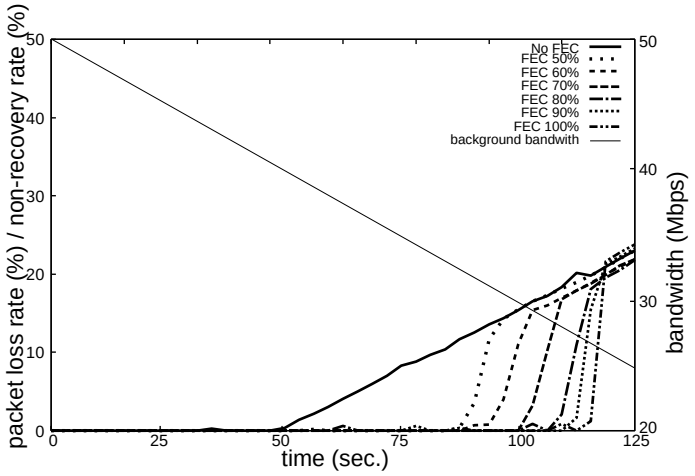


Figure A.3: **Packet loss and non-recovery rate in half rate DVTS.**

redundancy, we decided that the maximum FEC encoding rate in the half rate could be 100%. Note that if non-recovery rate is not converged even with 100% FEC redundancy with half rate transmission, DVTS continuously keeps the same condition.

A.3.4 Adaptive DVTS Implementation

In this section, we define an algorithm for the adaptive DVTS that dynamically adjusts transmission rate and FEC encoding rate in light of our NS-2 simulation and the experimental analysis aforementioned.

Table A.2 and A.3 provide the information how the flow type is changed during DV streaming. The algorithm checks the network state once every150 frame data (i.e. 5 seconds). For instance, on the condition with the full rate streaming with no FEC, if 1% packet loss is observed by the receiver and the number of consecutive loss is 5, then the application adds 20% FEC redundancy due to the network state that is defined "F5" in Table A.2. And if the network state in the next 150 frame transmission is not changed (i.e. F5 with FEC 20% in the table), then the FEC rate is decreased to 10%. This is because the sender recognizes available bandwidth has not decreased drastically and the streaming quality can be maintained even using lower FEC. If the network state in the next 150 frame transmission is changed to "F7", the FEC rate is increased to 30%. If the network

Table A.2: **Network states and flow types in full rate transmission (30 Mbps).**

state	P_{los}	N	0%	10%	20%	30%
F1	0	0	0%	0%	0%	0%
F2	$0 < P_{los} < 1$	$0 \leq N < 3$	10%	10%	10%	0%
F3		$3 \leq N < 10$	20%	10%	10%	10%
F4		$10 \leq N$	30%	20%	20%	10%
F5	$1 \leq P_{los} < 3$	$0 \leq N < 10$	20%	20%	10%	10%
F6		$10 \leq N < 25$	20%	20%	20%	20%
F7		$25 \leq N$	30%	30%	30%	30%
F8	$3 \leq P_{los} < 8$	$0 \leq N < 30$	20%	10%	10%	10%
F9		$30 \leq N < 70$	30%	20%	20%	20%
F10		$70 \leq N$	Half + 50%	30%	30%	30%
F11	$8 \leq P_{los} < 13$	$0 \leq N < 100$	30%	20%	20%	20%
F12		$100 \leq N < 170$	Half + 50%	30%	20%	20%
F13		$170 \leq N$	Half + 60%	Half + 50%	30%	30%
F14	$13 \leq P_{los} < 18$	$0 \leq N < 180$	Half + 60%	Half + 50%	30%	30%
F15		$180 \leq N < 250$	Half + 70%	Half + 60%	Half + 50%	Half + 50%
F16		$250 \leq N$	Half + 80%	Half + 70%	Half + 60%	Half + 60%
F17	$18 \leq P_{los} < 23$	$0 \leq N \leq 300$	Half + 80%	Half + 70%	Half + 60%	Half + 60%
F18		$300 \leq N < 380$	Half + 90%	Half + 80%	Half + 80%	Half + 70%
F19		$380 \leq N$	Half + 90%	Half + 90%	Half + 90%	Half + 80%
F20	$23 \leq P_{los}$	N = any	Half + 90%	Half + 90%	Half + 90%	Half + 90%

state is changed to "F15", the sender recognizes network congestion and changes the transmission rate to half and the FEC rate to 50%. The sender then uses the algorithm defined in Table A.3 for the next transmission. Our proposed mechanism feasibly increases the transmission rate when an application recognizes that the network becomes stable or the available bandwidth is increased (i.e. "H1" on the condition with 100% FEC). The mechanism changes the transmission rate back to the full rate without the extreme quality degradation, because FEC seamlessly recovers the data even if packet loss is again happened during expecting the available bandwidth. This mechanism has been the operational aspiration in the traditional rate control algorithm or quality adaptation mechanism. The algorithm shown in Table A.2 and A.3 is also adapted the condition in which FEC does not completely recover packet losses. When the FEC recovery rate is continuously lower than 80% with the same flow type within 450 frame data transmission, the algorithm increases 10% redundancy from the current FEC encoding rate, except the case that the FEC rate is already maximum in the transmission rate (i.e. 30% FEC with full rate transmission or 100% FEC with half rate transmission). If it happens on the condition with the 30% FEC with full rate transmission, the algorithm changes the flow type to 50% FEC with half rate transmission.

Table A.3: **Network states and flow types in half rate transmission (15 Mbps).**

state	P_{los}	N	50%	60%	70%	80%	90%	100%
H1	0	0	100%	100%	100%	100%	100%	Full + 0%
H2	$0 < P_{los} < 1$	$0 \leq N < 3$	100%	100%	100%	100%	100%	Full + 10%
H3		$3 \leq N < 10$	100%	100%	100%	100%	100%	Full + 20%
H4		$10 \leq N$	100%	100%	100%	100%	100%	Full + 30%
H5	$1 \leq P_{los} < 3$	$0 \leq N < 10$	100%	100%	100%	100%	100%	Full + 20%
H6		$10 \leq N < 25$	100%	100%	100%	100%	100%	Full + 20%
H7		$25 \leq N$	90%	100%	100%	100%	100%	Full + 30%
H8	$3 \leq P_{los} < 8$	$0 \leq N < 30$	90%	90%	90%	90%	100%	Full + 20%
H9		$30 \leq N < 70$	90%	90%	90%	90%	100%	Full + 30%
H10		$70 \leq N$	90%	90%	90%	80%	70%	60%
H11	$8 \leq P_{los} < 13$	$0 \leq N < 100$	90%	90%	90%	90%	100%	Full + 30%
H12		$100 \leq N < 170$	90%	90%	90%	70%	70%	60%
H13		$170 \leq N$	90%	90%	90%	60%	60%	60%
H14	$13 \leq P_{los} < 18$	$0 \leq N < 180$	60%	60%	60%	50%	50%	50%
H15		$180 \leq N < 250$	80%	80%	70%	70%	70%	70%
H16		$250 \leq N$	90%	90%	90%	80%	80%	80%
H17	$18 \leq P_{los} < 23$	$0 \leq N < 300$	90%	90%	90%	80%	80%	80%
H18		$300 \leq N < 380$	90%	90%	90%	90%	80%	80%
H19		$380 \leq N$	100%	100%	100%	90%	90%	90%
H20	$23 \leq P_{los} < 28$	$0 \leq N < 420$	100%	100%	90%	90%	90%	90%
H21		$420 \leq N < 600$	100%	100%	100%	90%	90%	90%
H22		$600 \leq N$	100%	100%	100%	100%	100%	100%
H23	$28 \leq P_{los} < 33$	$0 \leq N < 750$	100%	100%	100%	100%	90%	90%
H24		$750 \leq N$	100%	100%	100%	100%	100%	100%
H25	$33 \leq P_{los}$	$0 \leq N < 1300$	100%	100%	100%	100%	100%	90%
H26		$1300 \leq N$	100%	100%	100%	100%	100%	100%

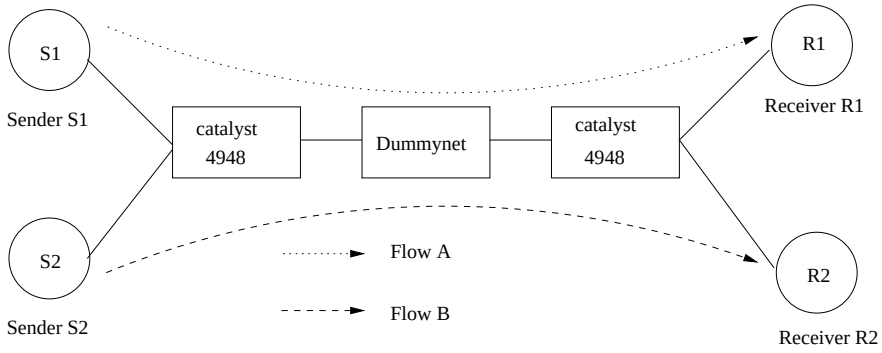


Figure A.4: **Testbed network for evaluating the implementation.**

A.4 Evaluation

To evaluate the effectiveness of the adaptive rate control mechanism, we set up *dummynet* and Cisco Catalyst switch boxes in the testbed network (Fig.A.4) with

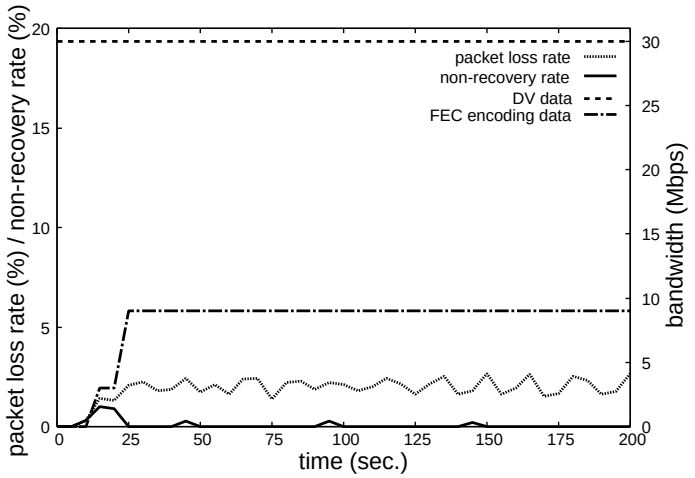


Figure A.5: **Competition with normal DVTS flow over 100 Mbps link.**

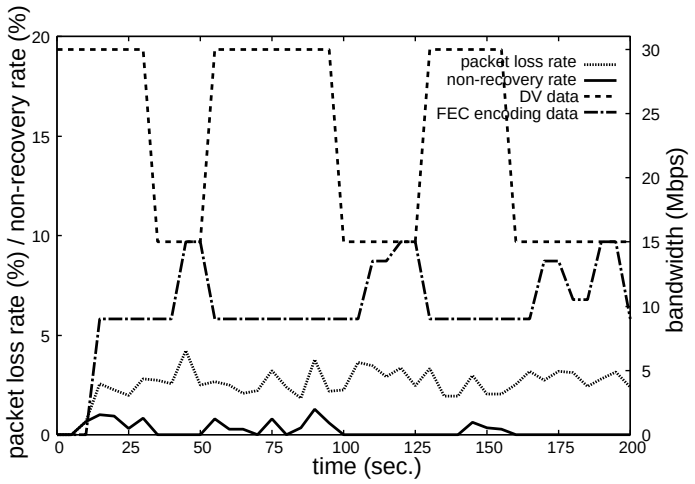


Figure A.6: **Competition with normal DVTS flow over 95 Mbps link.**

various configurations.

Fig.A.5, A.6, and A.7 show the measurement results obtained in our testbed network. In each test, S_1 transmits DV stream and R_1 receives it, and both hosts work with the adaptive DVTS. To add complexity, S_2 transmits another DV stream to R_2 in the middle of the evaluation to emulate network collision or congestion over shared link.

Fig.A.5 shows the behavior of the adaptive DVTS when the shared link band-

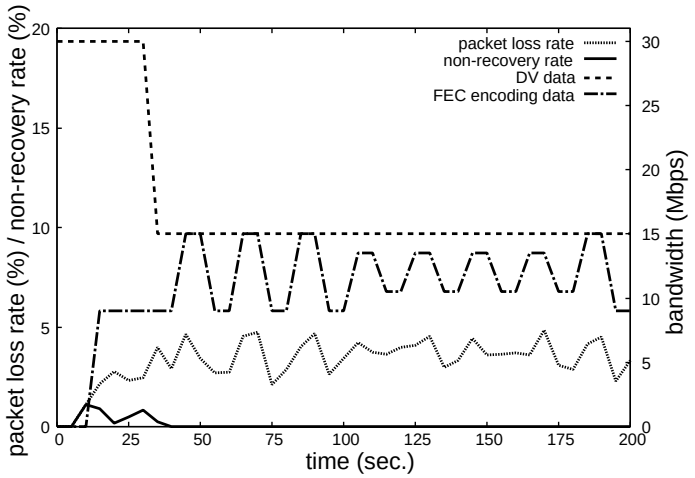


Figure A.7: **Competition with normal DVTS flow over 90 Mbps link.**

width was 100 Mbps. The DV stream consumed 30 Mbps bandwidth and the corresponding FEC redundancy used 9 Mbps (30% FEC). While the packet loss was observed in the transmission, non-recovery data was negligible (0.07%) after the flow type was converged. It took 17.4 sec to finally converge.

In Fig.A.6, we saw the rate control was repeated in the quality adaptation over 95 Mbps shared link, because our mechanism tries to keep high network utilization when possible. This was a worst case in the adaptive rate control mechanism, because quality degradation was observed (1% non-recovery data) in the full rate transmission.

Fig.A.7 shows the behavior of the adaptive DVTS over 90 Mbps shared link. The flow converged at the half rate transmission after the packet loss was observed. Then, to determine whether the transmission rate could be increased, our mechanism frequently changed the only FEC rate. In this phase, packet loss happened, but the quality degradation was avoided by FEC recovery. Although FEC caused 27.6 sec. delay of playing DV data in the receiver-side [73], the non-recovery rate was suppressed to 0.09%. It took 27.6 sec to converge at the half rate transmission.

To emulate network congestion and recovery, S_2 sent the full rate DV stream to R_2 over 90 Mbps shared link, then stopped. Fig.A.8 shows the behavior of our mechanism. In this measurement, when S_2's DVTS flow competed in the link (in 10 sec.), the adaptive DVTS increased the FEC rate and then decreased the

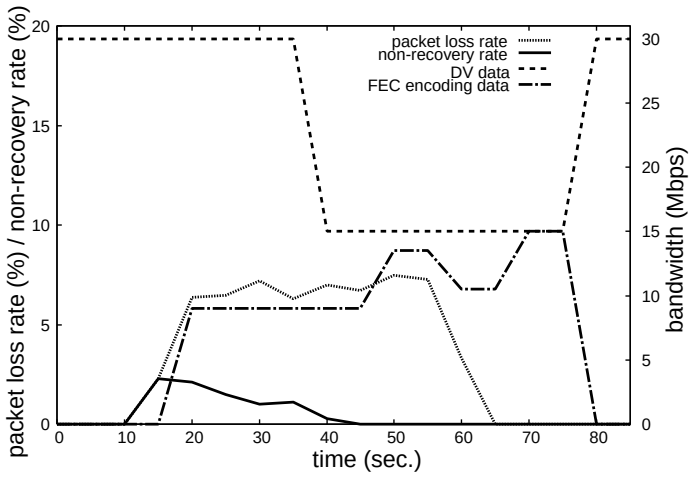


Figure A.8: **Adaptive DVTS in the recovery of the network congestion.**

transmission rate (35 sec.), because it observed consecutive packet loss. After S_2 stopped the flow, the packet loss was decreased (55 sec.). The adaptive DVTS adjusted the FEC rate to expect the available bandwidth (55 to 70 sec.), and then reduced the FEC rate and returned the full rate transmission. It took 17.8 sec to return to the full rate transmission.

A.5 Summary

In this appendix, we presented an *adaptive* rate control mechanism in DVTS that provides the best possible quality for DV streaming. It gives a good balance to incorporate transmission rate control and dynamic FEC technologies.

The adaptive DVTS controls the transmission rate and the FEC encoding rate based on the network condition between a sender and a receiver. It investigates packet loss rate, the number of consecutively lost packets, and FEC recovery rate during streaming. After the mechanism examines the condition of the network, it decides the flow type, which is defined as the combination of the data transmission rate and FEC encoding rate, and transmits the appropriate DV stream to the receiver.

We verified efficiency of the adaptive DVTS implementation on our testbed network. According to the result, our proposed mechanism utilizes network resource

effectively. It accordingly maintains DV streaming quality by dynamic FEC, even though packet loss occurs. However, we saw the transmission rate did not converge in the worst case with the quality degradation.